(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

B6

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 February 2003 (06.02.2003)

PCT

(10) International Publication Number
WO 03/010634 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/US02/23713

(22) International Filing Date: 26 July 2002 (26.07.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/912,408 26 July 2001 (26.07.2001) US

(71) Applicant: **PEGASUS COMMUNICATION GROUP** [US/US]; 225 City Line Avenue, Suite 200, Bala Cynwyd, PA 19004 (US).

(72) Inventors: **CASSIN, Lionel**; 835 N. 28th Street, Philadelphia, PA 19130 (US). **MILLER, Ronnen**; 835 N. 28th Street, Philadelphia, PA 19130 (US). **RENN, Luke**; 412 Main Street, Riverton, NJ 08077 (US). **PELLEGRINO, Donald**; 2201 Pennsylvania Avenue, Apartment 1310, Philadelphia, PA 19130 (US). **ACQUESTA, David**; 7818 Devon Street, Philadelphia, PA 19118 (US).

(74) Agents: **JACOBS, Leslie, Jr.** et al.; Arnold & Porter, 555 12th Street, NW, Washington, DC 20004 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 03/010634 A2

(54) Title: DEVICES, METHODS AND A SYSTEM FOR IMPLEMENTING A MEDIA CONTENT DELIVERY AND PLAYBACK SCHEME

(57) Abstract: Devices, methods and a system are provided for implementing a media content delivery and playback scheme, in which media content is delivered asynchronously via a communication channel to facilitate playback of the media content via a remote device.

DEVICES, METHODS AND A SYSTEM FOR IMPLEMENTING A MEDIA CONTENT DELIVERY AND PLAYBACK SCHEME

REFERENCE TO COMPACT DISC APPENDIX

A compact disc appendix including a computer program listing is filed herewith. The compact disc appendix includes the computer source code of a preferred embodiment of the present invention. Other embodiments of the present invention may be implemented using other computer code, using dedicated electronic hardware, using a combination of these, or otherwise. The contents of the compact disc appendix are incorporated herein in their entirety and are to be considered to be part of the disclosure of this specification, the files, dates of creation and size in bytes of each file are listed in Figures 11 – 23.

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates to the distribution of media content. In particular, the present invention relates to devices, methods and a system for implementing a media content delivery and playback scheme. The present invention provides for the delivery of media content asynchronously via a communication channel to facilitate playback of the media content through a remote device.

2. DESCRIPTION OF RELATED ART

A conventional system for the delivery of media content may utilize media streaming, a technique whereby media content is delivered to a remote device in

small segments. Each of the segments is stored in a buffer until there are a sufficient number of segments stored within the buffer to provide the user of the remote device with the opportunity to playback the media content in what appears to the user to be in a continuous stream. However, if there are problems with delivering the media content over the network, the playback of the media content may be disrupted. A conventional system is disclosed in U.S. Patent No. 5,917,835 to Progressive Networks, Inc., which is incorporated herein by reference. This system has been marketed under the trade name "REAL AUDIO"®.

SUMMARY OF THE INVENTION

The principal advantage of the present invention is the provision of devices, methods and a system for implementing a media content delivery and playback scheme.

According to a first embodiment of the present invention, a device is provided including a processor that controls asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device. The playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to one aspect of the first embodiment, the device may deliver the media content. According to another aspect of the first embodiment, the communication channel includes a satellite communication channel. According to yet another aspect of the first embodiment, the device may be a server

computer. According to still another aspect of the first embodiment, the delivery of the media content may be controlled in accordance with a digital rights management scheme. According to yet another aspect of the first embodiment, the remote device may include a client computer. According to yet another aspect of the first embodiment, the remote device may be a portable device. In addition, the portable device may be a wireless device, such as a cellular phone.

According to a second embodiment of the present invention, a device is provided including a processor that controls asynchronous delivery of media content over a communication channel including a satellite system to facilitate playback of the media content through a remote device. The playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to a third embodiment of the present invention, a device is provided including a processor executing software instructions including a software module. The software module includes a software delivery module that controls asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device. The playback of the media content is enabled at a first predetermined time after the delivery of the media content.

According to one aspect of the third embodiment, the device may deliver the media content. According to another aspect of the present invention, the device may be a server computer. According to another aspect of the present

invention, the device also includes a memory portion that stores at least a portion of the software module.

According to still another aspect of the third embodiment, the software delivery module generates indicator data for the remote device that provide an indication of a second predetermined time when the media content will be delivered to the remote device. The device delivers the media content to the remote device at the second predetermined time. According to still yet another aspect of the third embodiment, the remote device initiates a session with the software delivery module not prior to the second predetermined time. According to yet another aspect of the present invention, the remote device initiates the session by making a request for a connection with the device. The device may establish the connection in response to the request. The device then provides the remote device with an indication that a user of the remote device is entitled to the media content. The remote device then accepts the media content for delivery unless the remote device already has the media content.

According to yet another aspect of the third embodiment, the software module may also include a software recovery module, which provides control information to the software delivery module to enable the automatic delivery of disrupted data without delivering data that has already been successfully delivered to the remote device.

According to still yet another aspect of the third embodiment, the software module may also include a software database interface module that processes requests to retrieve information from a database including media content

information related to the media content. The software database interface module may receive a request for the information from the remote device, submit the request to the database, receive the information from the database, and send the information to the remote device. The software database interface module may also receive a request for the information from the software delivery module, submit the request to the database, receive the information, and send the information to the software delivery module to facilitate the delivery of the media content to the remote device.

According to yet another aspect of the third embodiment, the media content information may include at least an identifier identifying a media category with which the media content is associated. The media category may be a segment of an episode, an episode, a series, or a package with which the media content is associated. The package may be defined in accordance with user statistical information related to media usage by a user employing the remote device. The software delivery module may control the delivery of the media content based on user statistical information concerning media usage by a user employing the remote device.

According to yet another aspect of the third embodiment, the software delivery module may control the delivery of the media content in segments, each having a size which depends on the user statistical information.

According to still yet another aspect of the third embodiment, the software module may also include a software user interface module that processes requests for the information from a user of the remote device and submits the requests for

the information to the software database interface module for retrieval from the database. The software user interface module may include a graphical user interface. The graphical user interface may be implemented via a web site.

According to yet another aspect of the third embodiment, the software module may also include a software storage module that facilitates the storage of media content in a media content repository by a content provider.

According to still yet another aspect of the third embodiment, the software database interface module may receive a request for the information from the software storage module, submit the request to the database, receive the information from the database, and send the information to the software storage module to facilitate storage of the media content in the media content repository.

According to yet another aspect of the third embodiment, the software module may also include a software content provider interface module that processes requests for the information from a content provider and submits the requests for the information to the software database interface module for retrieval from the database. The software content provider interface module may include a graphical user interface. The graphical user interface may be implemented via a web site.

According to a fourth embodiment of the present invention, a device is provided that includes a processor that controls playback of media content delivered asynchronously over a communication channel by a remote device. The playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to one aspect of the fourth embodiment, the media content is not detectable by a user of the device until the predetermined time.

According to another aspect of the fourth embodiment, the processor may control the playback of media content via a display.

According to yet another aspect of the fourth embodiment, the device may include a display and the processor may control the playback of media content via the display.

According to still yet another aspect of the fourth embodiment, the device may be a computer, such as a client computer. In addition, the communication channel may include a network and the computer may be coupled to the remote device via the network.

According to still yet another aspect of the fourth embodiment, the device may be a portable device. The device may also be a wireless device, such as a cellular phone. In addition, the wireless device may include a display and the processor may control the playback of media content via the display.

According to a fifth embodiment of the present invention, a device is provided having a processor executing software instructions including a software module. The software module includes a first software playback module that controls the playback of media content delivered asynchronously over a communication channel by a remote device, wherein the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to a first aspect of the fifth embodiment, the media content is not detectable by a user of the device until the predetermined time.

According to another aspect of the fifth embodiment, the delivery of media content may be controlled in accordance with a digital rights management scheme.

According to another aspect of the fifth embodiment, the communication channel may include a satellite communication channel.

According to another aspect of the fifth embodiment, the first software playback module may control the playback of media content via a display. According to another aspect of the fifth embodiment, the device may include a display and the first software playback module controls the playback of media content via the display. According to still another aspect of the fifth embodiment, the first software playback module may include a graphical user interface through which the media content is displayed on the display.

According to another aspect of the fifth embodiment, the media content is not detectable by a user of the device until the predetermined time.

According to still yet another aspect of the fifth embodiment, the software module may also include a first software coordination module that coordinates the exchange of information with the remote device. The information includes the media content. In addition, the information may also include user statistical information related to media usage by a user employing the device. According to still another aspect of the fifth embodiment, the user statistical information may be sent by the device to the remote device to facilitate the delivery of the media content to the device.

According to another aspect of the fifth embodiment, the device may include a storage area that stores media data including the media content. The media data may include a number of media files, and the media content may be formed from a number of media files in accordance with at least one predefined rule. In addition, at least one of the number of media files may be used to form distinct media content.

According to still yet another aspect of the fifth embodiment the software module may also include a first registration module that receives user information from a user of the device. The device may also transmit the user information to the remote device to facilitate the delivery of the media content to the device.

According to another aspect of the fifth embodiment, the playback of media content is controlled based on user input. In addition, the user input may be provided to the device using a remote control device which communicates with the device. The remote control device may communicate with the device using infrared radiation.

According to still yet another aspect of the fifth embodiment, the software module may also include a voice recognition software module, which receives user input in the form of voice commands. The voice recognition software module converts the voice commands into electronic data and provides the first software playback module with the electronic data to facilitate the playback of media content.

According to sixth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered

asynchronously from a remote device. The device generates a notification for a user of the device upon receipt of the media content. The notification may be an automatic notification, an audio notification, or an e-mail, for example. According to another aspect of the sixth embodiment, the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to a seventh embodiment of the present invention, a device is provided including a processor that controls playback of media content delivered asynchronously over a communication channel including a satellite system by a remote device. The playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to an eighth embodiment of the present invention, a system is provided for implementing a media content delivery and playback scheme. The system includes a communication channel, a first device and a second device. The first device is coupled to the communication channel and includes a first processor that controls asynchronous delivery of media content over the communication channel. The second device is coupled to the communication channel and includes a second processor that controls the playback of media content delivered asynchronously over the communication channel by the first device, wherein the playback of media content is enabled in the second device at a first predetermined time after the delivery of the media content.

According to one aspect of the eighth embodiment, the media content is not detectable by a user of the second device until the predetermined time.

According to another aspect of the eighth embodiment, the second device may initiate a session with the first device at the predetermined time. The second device may initiate the session by making a request for a connection with the first device. The first device establishes the connection in response to the request. According to another aspect of the eighth embodiment, the first device provides the second device with an indication that a user of the second device is entitled to the media content. The second device may accept the media content for delivery only if it does not already have the media content.

According to another aspect of the eighth embodiment the first device may be a server computer and the second device may be a client computer.

According to another aspect of the eighth embodiment, the communication channel may include at least a portion of a network, such as a local area network or a wide area network. In addition, the communication channel may include at least a portion of the Internet.

According to another aspect of the eighth embodiment, the second device may be a portable device. The portable device may be a wireless device, such as a cellular phone.

According to another aspect of the eighth embodiment communication channel includes a wireless network.

According to another aspect of the eighth embodiment, the delivery of the media content from the first device to the second device is controlled in accordance with a digital rights management scheme.

According to a ninth embodiment of the present invention, a device is provided having a processor that controls the delivery of media content over a communication channel to a remote device in one of a first mode and a second mode. In the first mode, the processor controls the asynchronous delivery of media content over the communication channel to facilitate playback of the media content through the remote device. In the second mode, the processor controls the synchronous delivery of media content over the communication channel to facilitate the playback of the media content through the remote device.

According to a first aspect of the ninth embodiment, the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to another aspect of the ninth embodiment, the device delivers the media content.

According to another aspect of the ninth embodiment, the communication channel includes a satellite communication channel.

According to another aspect of the ninth embodiment, the device is a server computer.

According to another aspect of the ninth embodiment, the playback of the media content is controlled in accordance with a digital rights management scheme.

According to another aspect of the ninth embodiment, the remote device may include a client computer. The remote device may be a portable device, such as a wireless device. The wireless device may be a cellular phone.

According to a tenth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered over a communication channel by a remote device. The processor controls the playback of media content in one of a first mode and a second mode. In the first mode, the processor controls the playback of media content delivered asynchronously by the remote device. In the second mode, the processor controls the playback of media content delivered synchronously by the remote device.

According to a first aspect of the tenth embodiment, the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to another aspect of the tenth embodiment, the media content is not detectable by a user of the device until the predetermined time.

According to an eleventh embodiment of the present invention, a device is provided having a processor that controls the delivery of media content over a communication channel to a remote device in one of a first mode and a second mode. In the first mode, the processor controls the unicast-based delivery of media content over the communication channel to facilitate playback of the media content through the remote device. In the second mode, the processor controls the multicast-based delivery of media content over the communication channel to facilitate the playback of the media content through the remote device.

According to another aspect of the eleventh embodiment, the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to another aspect of the eleventh embodiment, the device may deliver the media content.

According to another aspect of the eleventh embodiment, the communication channel includes a satellite communication channel.

According to another aspect of the eleventh embodiment, the device may be a server computer.

According to another aspect of the eleventh embodiment, the playback of the media content may be controlled in accordance with a digital rights management scheme.

According to another aspect of the eleventh embodiment, the remote device may include a client computer. In addition, the remote device may be a portable device, such as a wireless device. The wireless device may be a cellular phone.

According to a twelfth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered over a communication channel by a remote device. The processor controls the playback of media content in one of a first mode and a second mode. In the first mode, the processor controls the playback of media content delivered by the remote device via a unicast mode of delivery. In the second mode, the processor controls the playback of media content delivered by the remote device via a multicast mode of delivery.

According to a first aspect of the twelfth embodiment, the playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to another aspect of the twelfth embodiment, the media content is not detectable by a user of the device until the predetermined time.

According to a thirteenth embodiment of the present invention, a device is provided having a processor that controls asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device. The device receives a request for a connection from the remote device, establishes the connection in response to the request, provides the remote device with a first indication that a user of the remote device is entitled to the media content, and receives from the remote device a second indication that the remote device will accept the media content for delivery unless the remote device already has the media content.

According to a first aspect of the thirteenth embodiment, the first indication includes a first list of a first group of media content items including at least a first media content item, which is the media content.

According to another aspect of the thirteenth embodiment, the second indication includes a second list of a second group of media content items including at least a second media content item, which is the media content. The second group of media content items includes a number of media content items including at least the second media content item. The second group of media

content items is a group of media content items that the remote device will accept for delivery from the device.

According to a fourteenth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered asynchronously over a communication channel by a remote device. The device makes a request for a connection to the remote device, receives a connection from the remote device in response to the request, receives a first indication from the remote device that a user of the device is entitled to the media content from the remote device, and provides a second indication to the remote device that the device will accept the media content for delivery unless the device already has the media content.

According to a first aspect of the fourteenth embodiment, the first indication includes a first list of a first group of media content items including at least a first media content item, which is the media content.

According to another aspect of the fourteenth embodiment, the second indication includes a second list of a second group of media content items including at least a second media content item, which is the media content. The second group of media content items includes a number of media content items including at least the second media content item. The second group of media content items is a group of media content items that the device will accept for delivery from the remote device.

According to a fifteenth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered

asynchronously over a communication channel by a remote device. The device is capable of providing an indication to another on behalf of a user of the device, the indication being of a location where the media content may be found.

According to a sixteenth embodiment of the present invention, a device is provided having a processor that controls playback of media content delivered asynchronously over a communication channel by a remote device. The device is capable of providing a portion of the media content to another on behalf of a user of the device. The portion of the media content may be provided as an attachment to an e-mail.

According to an seventeenth embodiment of the present invention, a computer program product is provided for use in a device having a processor for executing software instructions. The computer program product includes a computer usable medium having computer readable program code means embodied therein for causing the device to control the asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device. The playback of the media content is enabled at a first predetermined time after the delivery of the media content.

According to a eighteenth embodiment of the present invention, a computer program product is provided for use in a device having a processor for executing software instructions. The computer program product includes a computer usable medium having computer readable program code means embodied therein for causing the device to control playback of media content delivered asynchronously over a communication channel by a remote device. The

playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to a nineteenth embodiment of the present invention, a method of implementing a media content delivery and playback scheme is provided. The method includes the step of delivering media content asynchronously via a communication channel for remote playback of the media content. The remote playback of the media content is enabled at a predetermined time after the delivery of the media content.

According to a first aspect of the nineteenth embodiment, the media content is not detectable until the predetermined time.

According to another aspect of the nineteenth embodiment, the method also includes the steps of receiving the media content; and enabling the playback of the media content at the predetermined time.

According to another aspect of the nineteenth embodiment, the method also includes the step of conducting the playback of the media content after enabling the playback of the media content at the predetermined time.

According to another aspect of the nineteenth embodiment, the step of conducting may include the step of displaying the media content.

According to a twentieth embodiment of the present invention, a method of implementing a media content delivery and playback scheme is provided. The method includes the steps of receiving media content which is delivered asynchronously via a communication channel; and enabling playback of the media content at a predetermined time after the receipt of the media content.

According to a first aspect of the twentieth embodiment, the method includes the step of detecting the media content at the predetermined time.

According to another aspect of the twentieth embodiment, the method includes the step of providing a notification of receipt of the media content.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features, aspects and advantages of the present invention will become better understood with reference to the following description, appended claims, and accompanying drawings, in which:

Figure 1 depicts a system for implementing a media content delivery and playback scheme in accordance with an embodiment of the present invention.

Figure 2 depicts the components of a software module that may be employed in a first device in accordance with an embodiment of the present invention.

Figure 3 depicts the components of a software module that may be employed in a second device in accordance with an embodiment of the present invention.

Figure 4 depicts a flow chart featuring the steps of an embodiment of the present invention.

Figure 5 depicts another flow chart featuring the steps of another embodiment of the present invention.

Figure 6 depicts a system in accordance with a preferred embodiment of the present invention.

Figure 7a depicts an aspect of a graphical user interface of a software playback module employed in a client computer of the system depicted in **Figure 6.**

Figure 7b depicts another aspect of the graphical user interface shown in **Figure 7a.**

Figure 8 depicts a protocol for communications between a client computer and a server computer in the system depicted in **Figure 6.**

Figure 9 depicts another protocol for communications between a client computer and a server computer in the system depicted in **Figure 6.**

Figure 10 depicts classes of objects and their corresponding attributes for objects stored in a database of the system depicted in **Figure 6.**

Figures 11 through 23 depict the files, dates of creation, and size in bytes of the compact disc appendix.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is directed to devices, methods and a system for implementing a media content delivery and playback scheme. In particular, the present invention provides for the delivery of media content (including, for example, audio and video) asynchronously via a network to facilitate playback of

the media content through a remote device (*e.g.*, user-owned, client computer). The invention is implemented through the asynchronous delivery of media content by proactively storing media content to a remote device. However, the present invention may also support two modes of delivery: an asynchronous mode and a synchronous mode. Preferably, the delivery of media content is based on a user-generated content preference.

Figure 1 shows a system **100**, which implements a media content delivery and playback scheme in accordance with the present invention. The system **100** includes a first device **110** and a second device **120** coupled by a communication channel **130**. The first device **110** and the second device **120** may be implemented in computer systems (not shown), which include those devices **110** and **120**. The first device **110** is coupled to the communication channel **130** and includes a first processor **140** that controls asynchronous delivery of media content over the communication channel **130**. The second device **120** is coupled to the communication channel **130** and includes a second processor **150** that controls the playback of media content delivered asynchronously over the communication channel **130** by the first device **110** (or a device controlled by the first device **110**).

The playback of media content is enabled in the second device **120** at a first predetermined time after the delivery of the media content. In addition, the media content may not be detectable by a user of the second device **120** until the predetermined time. Preferably, the delivery of media content from the first

device **110** (or a device controlled by the first device **110**) to the second device **120** is controlled in accordance with a digital rights management scheme.

The devices **110** and **120** of the system **100** may, for example, operate in the following manner. The second device **120** may initiate a session with the first device **110** at the predetermined time. The second device **120** initiates the session by making a request for a connection with the first device **110**. The first device **110** establishes the connection in response to the request. The first device **110** provides the second device with an indication that a user of the second device **120** is entitled to the media content. The second device **120** may accept the media content for delivery only if it does not already have the media content.

In the system **100**, the first device **110** may be a server computer and the second device **120** may be a client computer. However, the system may be operated as a peer-to-peer system, in which either the first device **110** or the second device **120** operates as a server with respect to the other, which operates as the client. The first device **110** and/or the second device **120** may be a portable device. The portable device may be a wireless device, such as a cellular phone with or without a display.

The processor **150** of the second device **120** may control the playback of media content via a display. By way of example, the second device **120** may have associated with it a display **125** and the second processor **150** may control the playback of media content via the display **125**. Although the display **125** is shown as being part of the device **120**, the display **125** may be coupled to the device **120**.

The communication channel **130** may include at least a portion of a network, such as a local area network, wide area network, public network (e.g., the Internet), a wireless network, or a combination of networks. The communication channel **130** may also include a satellite communication channel, including a satellite communication system. If the communication channel **130** includes at least part of a network, the first device **110** and the second device **120** may be coupled to each other via the network. In addition, if the communication channel **130** utilizes a satellite communication system, a satellite communication system may be used in the system **100** which is similar to those featured in U.S. Patent No. 6,016,388 and U.S. Patent No. 6,205,473, which are incorporated herein by reference.

As noted above, the first device **110** includes a processor **140** that controls asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device (*e.g.*, the second device **120**). The processor **140** executes software instructions, which, in accordance with one embodiment of the present invention, may include a software module **160**, as shown in **Figures 1 and 2**. The first device **110** may also include a memory portion **170** that stores at least a portion of the software module.

The software module **160** includes a software delivery module **200** that controls asynchronous delivery of media content over a communication channel to facilitate playback of the media content through the remote device. The playback of the media content is enabled at a first predetermined time after the delivery of the media content.

The software delivery module **200** may, for example, generate indicator data for the remote device that provide an indication of a second predetermined time when the media content will be delivered to the remote device. The first device **110** delivers the media content to the remote device at the second predetermined time. As noted above, the remote device may initiate a session with the software delivery module just prior to the second predetermined time. The remote device initiates the session by making a request for a connection with the first device **110**. The first device **110** may establish the connection in response to the request. The first device **110** then provides the remote device with an indication that a user of the remote device is entitled to the media content. The remote device then accepts the media content for delivery unless the remote device already has the media content.

The software module **160** may also include a software recovery module **210**, which provides control information to the software delivery module **200** to enable the automatic delivery of disrupted data without delivering data that has already been successfully delivered to the remote device.

The software module **160** may also include a software database interface module **220** that processes requests to retrieve information from a database (not shown) including media content information related to the media content. The software database interface module **220** may receive a request for the information from the remote device, submit the request to the database, receive the information from the database, and send the information to the remote device. The software database interface module **220** may also receive a request for the

information from the software delivery module **200**, submit the request to the database, receive the information, and send the information to the software delivery module **200** to facilitate the delivery of the media content to the remote device.

The media content information may include at least an identifier identifying a media category with which the media content is associated. The media category may be a segment of an episode, an episode, a series, or a package with which the media content is associated. The package may be defined in accordance with user statistical information related to media usage by a user employing the remote device. The software delivery module **200** may control the delivery of the media content based on user statistical information concerning media usage by a user employing the remote device. The software delivery module **200** may control the delivery of the media content in segments, each having a size that depends on the user statistical information.

The software module **160** may also include a software user interface module **230** that processes requests for the information from a user of the remote device and submits the requests for the information to the software database interface module **220** for retrieval from the database. The software user interface module **230** may include a graphical user interface **240**. The graphical user interface **240** may be implemented via a web site.

The software module **160** may also include a software storage module **250** that facilitates the storage of media content in a media content repository (not shown) by a content provider. The software database interface module **220** may

receive a request for the information from the software storage module **250**, submit the request to the database, receive the information from the database, and send the information to the software storage module **250** to facilitate storage of the media content in the media content repository.

The software module **160** may also include a software content provider interface module **260** that processes requests for the information from a content provider and submits the requests for the information to the software database interface module **220** for retrieval from the database. The software content provider interface module **260** may include a graphical user interface **270**. The graphical user interface **270** may be implemented via a web site.

The software module **160** described above in connection with the first device **110** may be stored on a computer program product in accordance with the present invention. By way of example, the computer program product includes a computer usable medium having computer readable program code means embodied therein for causing the first device **110** to control the asynchronous delivery of media content over a communication channel to facilitate playback of the media content through a remote device (e.g., the second device **120**). The playback of the media content is enabled at a first predetermined time after the delivery of the media content.

As noted above, the second device **120** includes a processor **150** that controls playback of media content delivered asynchronously over a communication channel by a remote device (e.g., the first device **110**). The processor **150** executes software instructions, which, in accordance with one

embodiment of the present invention, may include a software module **180**, as shown in **Figures 1 and 3**. The second device **120** may also include a memory portion **190** that stores at least a portion of the software module **180**.

The software module **180** includes a first software playback module **300** that controls the playback of media content delivered asynchronously over a communication channel by a remote device, wherein the playback of the media content is enabled at a predetermined time after the delivery of the media content. In addition, the media content may not be detectable by a user of the second device **120** until the predetermined time.

The media content may be distributed from the second device to another device (e.g., portable device). The distribution of the media content may be controlled in accordance with a digital rights management scheme, as set forth below.

The first software playback module **300** may control the playback of media content via a display. As noted above, the second device **120** may include a display **125** and the first software playback module **300** may control the playback of media content via the display **125**. The first software playback module **300** may include a graphical user interface **310** through which the media content is displayed on the display **125**.

The software module **180** may also include a first software coordination module **315** that coordinates the exchange of information with the remote device. The information includes the media content. In addition, the information may also include user statistical information related to media usage by a user employing the

second device **120**. The user statistical information may be sent by the second device **120** to the remote device to facilitate the delivery of the media content to the second device **120**.

The second device **120** may include a storage area (not shown) that stores media data including the media content. The media data may include a number of media files, and the media content may be formed from a number of media files in accordance with at least one predefined rule. In addition, at least one of the number of media files may be used to form distinct media content.

The software module **180** may also include a first software registration module **318** that receives user information from a user of the second device **120**. The second device **120** may also transmit the user information to the remote device to facilitate the delivery of the media content to the second device **120**.

The playback of media content is controlled based on user input. In addition, the user input may be provided to the second device **120** using a remote control device (not shown) that communicates with the device. The remote control device may communicate with the device using infrared radiation in a manner well known in the art.

The software module **180** may also include a voice recognition software module **320**, which receives user input in the form of voice commands. The voice recognition software module **320** converts the voice commands into electronic data and provides the first software playback module **300** with the electronic data to facilitate the playback of media content. Several voice recognition techniques which are known to those skilled in the art, may be implemented in the voice

recognition software module **320**. Examples of such voice recognition techniques are featured in U.S. Patent No. 6,094,635, U.S. Patent No. 6,154,722, and U.S. Patent No. 6,260,012, which are incorporated herein by reference.

The software module **180** described above in connection with the second device **120** may be stored on a computer program product in accordance with present invention. By way of example, the computer program product may include a computer usable medium having computer readable program code means embodied therein for causing the second device **120** to control playback of media content delivered asynchronously over a communication channel by a remote device (e.g., the first device **110**). The playback of the media content is enabled at a predetermined time after the delivery of the media content.

As an alternative to the second device **120**, a third device (not shown) may be employed in the system **100**. The third device includes a processor that controls playback of media content delivered asynchronously from a remote device. In addition, the third device generates a notification for a user of the device upon receipt of the media content. The notification may be an automatic notification, an audio notification, or an e-mail, for example. As with the second device, the playback of the media content may be enabled at a predetermined time after the delivery of the media content. Similarly, the media content may not be detectable by a user of the third device until the predetermined time.

The processors **140** and **150** of the first device are capable of operating in multiple modes, as described below. The processor **140** of the first device **110** may control the delivery of media content over a communication channel **130** to a

remote device (*e.g.*, the second device **120**) in one of a first mode and a second mode. In the first mode, the processor **140** controls the asynchronous delivery of media content over the communication channel **130** to facilitate playback of the media content through the remote device. In the second mode the processor **140** controls the synchronous delivery of media content over the communication channel **130** to facilitate the playback of the media content through the remote device.

The processor **140** of the first device **110** may control the delivery of media content over the communication channel **130** to a remote device (e.g., the second device **120**) in one of a first mode and a second mode. In the first mode, the processor **140** controls the unicast-based delivery of media content over the communication channel **130** to facilitate playback of the media content through the remote device. In the second mode, the processor **140** controls the multicast-based delivery of media content over the communication channel **130** to facilitate the playback of the media content through the remote device.

A synchronous mode of delivery refers to the delivery of media content in a manner in which the media content is being played back (i.e., watched or listened) at nearly the same time it is being delivered. With respect to a communication channel **130**, such as a digital network or the Internet, digital data is stored momentarily in memory buffers before being played back. This mode of delivery is typically referred to as media streaming and the contents of the memory buffers do not survive the playback session. In contrast, an asynchronous delivery mode refers to the delivery of robust media content files

which survive the playback session. In this regard, the media content files are delivered "asynchronously" - i.e., **without respect to time**. Both the unicast and multicast–based mode of delivery are synchronous delivery nodes.

A unicast mode of delivery refers to point-to-point interactive communication over the communication channel **130. (e.g.,** the Internet). The first device **110** (*e.g.*, a server computer) and the second device **120** (e.g., a client computer) are holding a private and interactive session with each other. In this manner, the second device **120** can request a specific media content item, and the first device **110** delivers that media content item solely to the second device **120.** In a multicast mode of delivery, the first device **110** broadcasts the media content items to all devices (including the second device **120**) in the communication channel **130.** The only way that the first device **110** can control which devices can utilize those media content items is to utilize a digital rights management scheme. By way of example, the media content items may be encrypted and encryption keys distributed to specific devices. The encryption keys are specifically matched to a unique receiver ID. This is the basis for conventional conditional access systems that are used to control cable and satellite TV access in a manner well known in the art.

The processor **150** of the second device **120** may control the playback of media content delivered over the communication channel **130** by a remote device (*e.g.*, the first device **110**). The processor controls the playback of media content in one of a first mode and a second mode. In the first mode, the processor

controls the playback of media content delivered asynchronously by the remote device. In the second mode, the processor controls the playback of media content delivered synchronously by the remote device.

The processor **150** of the second device **120** controls the playback of media content delivered over the communication channel **130** by a remote device (*e.g.*, the first device **110**). The processor **150** controls the playback of media content in one of a first mode and a second mode. In the first mode, the processor **150** controls the playback of media content delivered by the remote device via a unicast mode of delivery. In the second mode, the processor **150** controls the playback of media content delivered by the remote device via a multicast mode of delivery.

The processor **140** of the first device **110** may control the asynchronous delivery of media content, as follows. The first device **110** receives a request for a connection from a remote device (e.g., the second device **120**). Thereafter, the first device **110** establishes the connection in response to the request of the remote device and provides the remote device with a first indication that a user of the remote device is entitled to the media content. Then, the first device **110** receives from the remote device a second indication that the remote device will accept the media content for delivery unless the remote device already has the media content.

The first indication may include a first list of a first group of media content items including at least a first media content item, which is the media content. The second indication may include a second list of a second group of

media content items including at least a second media content item, which is the media content. The second group of media content items includes a number of media content items including at least the second media content item. The second group of media content items is a group of media content items that the remote device will accept for delivery from the device.

The processor **140** of the second device **120** may control the playback of media content delivered asynchronously over the communication channel **130** by a remote device (*e.g.*, the first device **110**) as follows. The second device **120** makes a request for a connection to the remote device. Thereafter, the second device **120** receives a connection from the remote device in response to the request. The second device **120** receives a first indication from the remote device that a user of the second device **120** is entitled to the media content from the remote device. Then, the second device **120** provides a second indication to the remote device that the second device **120** will accept the media content for delivery unless the second device **120** already has the media content.

The first indication may include a first list of a first group of media content items including at least a first media content item, which is the media content. The second indication may include a second list of a second group of media content items including at least a second media content item, which is the media content. The second group of media content items includes a number of media content items including at least the second media content item. The second group of media content items is a group of media content items that the second device **120** will accept for delivery from the remote device.

Two additional alternatives for the second device **120** will now be described to highlight two additional features of the present invention. In one of the additional alternatives, a fourth device (not numbered) is provided having a processor (not numbered) that controls playback of media content delivered asynchronously over a communication channel by a remote device. The device is capable of providing an indication to another on behalf of a user of the device, the indication being of a location where the media content may be found. By way of example, the indication may be a URL address. In another of the additional alternatives, a fifth device is provided having a processor that controls the playback of media content delivered asynchronously over a communication channel by a remote device. The device is capable of providing a portion of the media content to another on behalf of a user of the device. The portion of the media content may be provided as an attachment to an e-mail.

Figure 4 shows a flow chart featuring the steps of a method of implementing a media content delivery and playback scheme in accordance with the present invention. The method includes the step **400** of delivering media content asynchronously via a communication channel for remote playback of the media content. The remote playback of the media content is enabled at a predetermined time after the delivery of the media content. In addition, the media content may not be detectable until the predetermined time. In step **410**, media content is received. Thereafter, in step **420**, the playback of the media content is enabled at the predetermined time. The method may also include the step **430** of conducting the playback of the media content after enabling the playback of the

media content at the predetermined time. The step of conducting may include the step (not shown) of displaying the media content.

Figure 5 shows a flow chart featuring the steps of a method of implementing a media content delivery and playback scheme in accordance with the present invention. The method includes the step **500** of receiving media content which is delivered asynchronously via a communication channel, and the step **510** of enabling the playback of the media content at a predetermined time after the receipt of the media content. The method may also include the step **503** of providing a notification of the receipt of the media content. In addition, the method may also include the step (not shown) of detecting the media content at the predetermined time, in which case, the notification Step **503** will not occur until the media content is detected.

Figure 6 depicts a system **600** for implementing a media content delivery and playback scheme in accordance with a preferred embodiment of the present invention. The system includes a server computer system **610** including at least a server computer (not shown) having a first processor (not shown). The system also includes a client computer system **620** including at least a client computer (not shown) having a processor (not shown). In addition, the system **600** includes at least a portion of a network **630** by which the server computer system **610** and the client computer system **620** are coupled to each other. The network **630** may be implemented as a local area network, wide area network, a public access network (*e.g.*, the Internet), or a combination of networks.

Both the server and client computers may be implemented as a portable device (*e.g.*, a personal digital assistant), a wireless device, or a portable wireless device (*e.g.*, cellular phone or pager). In addition, although the system **600** is described as a server/client based system, it may also be arranged as a peer-to-peer system, in which each device acts as server with respect to the other device, which, in turn, acts as a client.

The processor of the server computer executes instructions including a first software module **632**, which may be stored in a storage device associated with the server computer, or on another device with which the server computer is networked. The storage device may include a hard drive, random access memory, read only memory, a redundant array of inexpensive disks (RAID), an optical disk, a CD-ROM, WORM, floppy disk, or any of a number of storage devices, which are well known to those skilled in the art.

The first software module **632** includes a software delivery module **635** that controls asynchronous delivery of media content over the network **630** to facilitate playback of the media content through the client computer. By storing the media content on the client computer (*i.e.*, a remote device), which is local to the user of the client computer, the system **600** avoids the inconsistent quality that may result from streaming media content over the network **630**. By downloading media content in the form of media files automatically, the system **600** eliminates the tedium of hunting for media files and manually downloading them. In addition, the playback of the media content may, in accordance with one aspect of the preferred embodiment, only be enabled at a predetermined time after the

delivery of the media content. In addition, the media content may not be detectable by a user of the client computer until the predetermined time.

Preferably, the software delivery module **635** waits to be contacted by the client computer and then interacts with the client computer to coordinate the delivery of media content to the client computer and to coordinate the receipt of user statistical information back from the client computer.

Since the system **600** relies on the download of media content to an often unattended client computer, there may be media content items that are downloaded to the client computer that a user of the client computer will never use. Thus, if the server computer delivered media content items in their entirety, the useable bandwidth for the network **630** may not be allocated efficiently. The first software module **632** may include an adaptive download module (not shown) that monitors the user's actual consumption patterns and determines the media content items that have a higher probability of being consumed or of not being consumed. For those media content items that do not have a high probability of being consumed, the adaptive download module would only download portions of those content items or perhaps cease downloading them altogether. Thus, the total bandwidth load for the network **630** would be reduced.

The server computer may incorporate a database **640** (*e.g.*, a PostgreSQL relational database), or it may have access to the database **640**, which may be residing on a storage device within the server computer system **610**, or which may be otherwise accessible via the network **630**. The database **640** stores information related to the system in the form of metadata. The information must include

metadata about the media content available to the system, including for example, descriptions of packages, series and episodes. The information should also include metadata that relates to users of the system. This metadata may include, for example, account information, billing history and statistics. The database need not contain the media files themselves. As an alternative, the media files may be stored within a first content repository **645** associated with the server computer system **610**. The contents of the first content repository **645** may be accessible via the database **640**.

The first software module **632** may also include a software user interface module **650** and a software content provider interface module **655**. These are the public interfaces for users, advertisers and content providers. These interfaces may be operated as graphical user interfaces that are implemented as web sites or web portals. By way of example, users would use the software user interface module **650** to change their subscriptions, view billing histories, or view the available media content (as represented by, for example, episodes, shows, series or pre-defined packages). In addition, the content provider interface module **655** may be accessed by a content provider via a web browser **657** residing on a client computer **658** of the content provider.

The software interface module **650** may also be implemented with a program guide, which would allow users of the client computer to preview media content items. The program guide would allow the user to preview selected media content items by streaming those items (*e.g.*, audio or video content items) to the client computer. A clickable hyperlink in the program guide will launch the

user's streaming media player of choice and allow them to read, listen, or see the media content item before subscribing. In addition, content providers may use a software content provider interface module **655**, to upload new content and view summary statistics about how their shows were being used by users of the system **600**.

The processor of the client computer executes instructions including a second software module which may be stored in a storage device associated with the client computer, or on another device with which the client computer is networked. The storage device may include a hard drive, random access memory, read only memory, a redundant array of inexpensive disks (RAID), an optical disk, a CD-ROM, WORM, floppy disk, or any of a number of well-known storage devices, which are well known to those skilled in the art.

The second software module **659** includes a first software playback module **660** that controls the playback of media content delivered asynchronously over the network **630** by the server computer. The first software playback module **660** includes a graphical user interface **700**, as shown in **Figure 7a**, that allows the user to view, organize and play back media content. In particular, the graphical user interface **700** of the first software playback module **660** may include various windowpanes. By way of example, there may be a Folders Pane **705** that displays various folders for storing and organizing messages (*e.g.*, an Inbox folder **710**, a Saved folder **715** and a Trash folder **717**). There may also be a List Pane **720** that lists the media content items contained in a selected folder. By way of example, the List Pane **720** may identify the Show, Episode number,

Description, Receive Date and Publish Date of the media content items in the Inbox folder **710**. In addition, there may also be a Detail Pane **730** that displays detailed information about a media content item highlighted in the List Pane **720**. A user can move media files by dragging those media files and dropping them into specific folders.

The second software module also includes a first software coordination (or synchronization) module **665** that coordinates the exchange of information with the server computer. The first software coordination module **665** contacts the server computer and coordinates the delivery of any new media content files for a user of the client computer. The first software coordination module **665** also uploads user statistics back to the server computer. Using a client computer in accordance with the present invention, a user can have access to a wide variety of media content, including original and re-purposed data, music, videos and multi-media programming.

A variety of types of programming may be supported using the system **600** of the present invention. These types of programming include, for example, stock reports, news items, emergency reports, cartoons, movies, data reports, product reports and detailing, talk shows, music programs, do-it-yourself and repair information, horoscopes, audiobooks, news information, sports information, weather information, political information, dramas, NASCAR shows, personal relationship information and business reporting. The media content may also contain advertisements. In addition to or as an alternative to the use of advertisements, media content may be provided on a fee for content basis.

The second software module **659** also includes a software configuration module **670**. A user of the client computer can use the configuration module **670** to configure the software module, as desired.

The second software module also includes a software registration module (not specifically shown) that receives user information from a user of the client computer. The registration module may be implemented as a web browser **680** through which a user of the client computer can interact with the server computer to register or obtain information about media content or the user's account status. The client computer then transmits the user information to the server computer to facilitate the delivery of media content to the client computer.

The second software module may also access a local content repository **685** to store media content in the form of media files. The local content repository **685** may be a storage device, such as a hard drive, random access memory, a redundant array of inexpensive disks (RAID), an optical disk, a CD-RW, WORM, floppy disk, or any of a number of storage devices that are well known to those skilled in the art.

Initially, a new user will register with the system **600** using the web browser **680**. The new user registration process is a one-time event for each user. Using the web browser **680**, a new user will access the server computer's software user interface module **650**, which is implemented as a web site (or web portal) for end-users. When accessing the server client's web site, the user is prompted to become a system user in accordance with a registration process. In particular, the user fills out an on-screen HTML or XML form (not shown) with various pieces

of information including, for example: the user's name, address, e-mail address, credit card information, etc. The server computer uses this information to uniquely identify each of its users to: (a) ensure that the user gets the content they requested; (b) ensure that an administrator of the system gets paid for the services it renders; and (c) provide the system with valuable information for further use (e.g., e-mail addresses for new show notifications).

Once registered, a user would next receive the second software module **659**. The user may receive the second software module **659** by downloading it from the server computer's software user interface module **650** to the client computer. A variant on this process would be to distribute the second software module to the user on a storage device, such as a CD-ROM. Thus, instead of downloading the second software module from the server computer's user interface module **650**, a user would install the second software module from the storage device. A user would agree to the terms and conditions of use for the second software module and install it on the client computer.

Once the user was registered and had the second software module installed on the client computer, the user would access the server computer's software user interface module **650** via the web browser **680** and identify the media content that the user wishes to receive. This would occur by selecting content from the software user interface module **650**. The resulting user selection profile is stored in the database **640**.

Encoded in the second software module is the address (*e.g.*, IP address) for the server computer. The client computer uses this address to contact the

server computer and establish a connection. Once the connection is made, the client sends information that uniquely identifies that user.

The client computer may contact the server computer on a fixed time interval (e.g., every 15 minutes). Further, the server computer can also provide the client computer with an indication of a predetermined time when the client computer can expect the next piece of content to arrive. This later technique is particularly efficient at handling "special bulletins" and other content that is made available outside of a regular schedule of programming, for example.

Once the client computer identifies the user to the server computer, the server computer takes that information and queries the database **640** for all content to which that user is subscribed. The server computer then sends this entire list to the client. This is done so that the client computer can display on the graphical user interface **700** of the first software playback module **660** a progress bar, which indicates how many items have been received and how many are yet to come.

The server computer then proceeds to offer every one of these content items to the client computer for download. The client computer checks to see if it already possesses the specific media content in the form of a media file. If the client computer possesses the specific media content item, then the client computer refuses the download offer and the server computer skips to the next item on the list. If the client computer does not have the media content item, then the client computer accepts the download of the media content item from the server computer and stores the media content in the local content repository **685**.

Although generally described above, the client and server computers may interact in accordance with one of two protocols. The first protocol is illustrated in **Figure 8**. In step **800**, the client computer contacts the server computer and requests a connection. In step **810**, the server computer accepts and establishes the connection. Thereafter, in step **820**, the client computer sends user information to the server. Thereafter, in step **825**, the server computer uses the user information to query the database **640**. In step **830**, the database responds with a list of all content to which that user is entitled. In step **840**, the server computer sends the list to the client. Thereafter, in step **850**, the server computer attempts to send a first media content item on the list to the client computer. In step **860**, the client computer determines if it already has the media content item stored in the local content repository **685**. If the client computer does have the media content item, then in step **870**, the client computer provides an indication to the server computer that it currently has the media content item, such that the server computer will offer the next media item on the list. If the client computer does not have the media content item, then in step **880**, the client computer accepts the media content item from the server computer. Thereafter, steps **850-880** are repeated for each media content item, as necessary.

After the media content item has been successfully downloaded, in step **880**, the step **885** may be performed as an additional option, in which the server computer stores in the database **640** an indication that the media content item has been successfully downloaded to the client computer. Thus, when the server queries the database about which items to offer for future download to the client

computer, the database **640** will return only those items which have not already been downloaded to the client computer, rather than a comprehensive list of all media content items to which a user of the client computer may be entitled.

The second protocol is illustrated in **Figure 9**. In step **900**, the client computer contacts the server computer and requests a connection. In step **910**, the server computer accepts and establishes the connection. Thereafter, in step **920**, the client computer sends user information to the server. In step **925**, the client computer would request a list of content that should be delivered. In step **927**, the server computer uses the user information to query the database **640**. In step **930**, the database **640** responds with a first list of all content to which that user is entitled, which may be implemented as, for example, XML file. In step **940**, the server computer sends the first list to the client computer. Thereafter, in step **945**, the client computer identifies those media content items on the first list that it does not already have in the local content repository **685**. In step **947**, the client computer would send a second list of only those media content items contained in the first list that it currently does not have stored in the local content repository **685**. In step **950**, the server computer delivers those media content items contained in the second list to the client computer.

After the media content items have been successfully delivered, in step **950**, the step **960** may be performed as an additional option, in which the server computer stores in the database **640** an indication that the media content item has been successfully downloaded to the client computer. Thus, when the server queries the database about which items to offer for future download to the client

computer, the database **640** will return only those items which have not already been downloaded to the client computer, rather than a comprehensive list of all media content items to which a user of the client computer may be entitled.

Once a media content item has been successfully downloaded to the client computer's local content repository **685**, the new media content item will appear in the graphical user interface **700** of the first software playback module **660**. In particular, the media content item will appear in the Inbox folder **710** of the graphical user interface **700** along with other media content items that have already been downloaded from the server computer.

Typically, a user would start by checking their Inbox folder **710** for new media content. The graphical user interface **700** of the first software playback module **660** displays new, unused content items in the List Pane **720** in highlighted text, and read/listened/viewed items in normal un-highlighted text. If a user clicked a single time on a particular media content item appearing in the List Pane **720** using a mouse associated with the client computer, details about the selected media item would appear in the Detail Pane **730**. To delete a media content item, a user would either drag the item using a mouse to the Trash folder **717**, or select the content item and press the DELETE key on a keyboard associated with the client computer. In addition, users of the client computer can manage media content items by creating folders in the Folders Pane **705** and then dragging and dropping media content items into those folders.

To playback a media content item (e.g., read, listen, or view a media content item), a user would double click on it in the List Pane **720** using a mouse.

This action would initiate a separate media playback window appearing in the graphical user interface **700** with discrete control functions (e.g., "PLAY," "STOP," "PAUSE," "FORWARD," "REVERSE," and "RECORD" control functions). In addition, as noted above, the discrete control functions may be activated with a remote control device, in a manner well known in the art. By way of example, a remote control device employing an infrared wave may be used.

Every time a user clicks one of the discrete control functions using a mouse or activates one of those functions using a remote control device, that action is recorded to a statistics log file stored on a storage device (*e.g.*, hard drive) associated with the client computer. By way of example, the log file may record a user/player ID, a content ID, the absolute time (AM/PM), and the offset from the beginning of the content file to the action (e.g., STOP or PLAY). At the end of every session between the client and server computers, the server computer queries the client computer as to whether a log file is present. If there is a log file present, it is uploaded to the server computer, where it is parsed and placed in the database **640** for future analysis. When the log file has been successfully uploaded, it is deleted off of the storage device associated with the client computer.

Figure 10 depicts classes of objects stored in the database **640** along with their attributes. These classes include a pc_session class **1000**, a pc_user class **1010**, a pc_player class **1020**, pc_delivery class **1030**, a pc_subscription class

1040, a pc_lisent class 1050, a pc_content class 1060, a pc_blurb class 1070, a pc_series class 1080, a pc_episode class 1085, and a pc_segment class 1090.

The pc_session class 1000 includes the following attributes: sessionid, userid, sessionstamp and active (a status indicator). The pc_user class 1010 includes the following attributes: userid, loginname, pwd (password), zip and lastlogin. The pc_player class 1020 includes the following attributes: playerid, userid, datecreated. The pc_delivery class 1030 includes the following attributes: deliveryid, playerid, segmentid, contentid, delivered (a status indicator). The pc_subscription class 1040 includes the following attributes: seriesid, userid, datecreated, and dateterminated. The pc_lisent class 1050 includes the following attributes: listenid, playerid, userid, contentid, segmentid, starttimestamp, stoptimestamp, startoffset, and stopoffset. The pc_content class 1060 includes the following attributes: contentid, filename, name, active (a status indicator), and segmentid. The pc_blurb class 1070 includes the following attributes: blurbid, contentid, startoffset, stopoffset, and description. The pc_series class 1080 includes the following attributes: active (a status indicator) and name. The pc_episode class 1085 includes the following attributes: episodeid, seriesid, name, active (a status indicator), sequence, and timecreated. The pc_segment class 1090 includes the following attributes: segmentid, episodeid, contentid, sequence, and description.

In addition to those features already mentioned there are additional features that may be associated with the second software module, as set forth below. By way of example, the second software module would provide a user of

the client computer with the capability to place a media content marker (or bookmark) within the content item to mark a place to which they want to return. This content marker feature could be implemented using additional control functions on the graphical user interface **700**. The functions would enable a user to create, rename, delete and forward such media content markers.

An example of another feature of the second software module is that it may generate a notification to a user of the client computer upon receipt of a media content item. The notification may be an automatic notification, an audio notification, or an e-mail, for example, which is sent to the user's e-mail address.

An example of another feature of the second software module is a locator indication feature, which would allow users of the client computer to send to anyone via the network an indication (e.g., an e-mail) of a location (i.e., link to a URL address) where the media content item may be found. At such a location, a recipient of the indicator may find not just media content items, but previews of the media content item featured using a media streaming technique.

An example of yet another feature of the second software module is a media content sharing feature, which would allow a user to send a portion or segment of a media content item to another individual. By way of example, a user could snip out a small segment of the content and attach that media content portion or segment directly to an e-mail for distribution to another individual. That individual would then be able to play this directly from the e-mail.

The distribution of media content via the media content sharing feature would be controlled using a digital rights management scheme as described

below. The digital rights management scheme would take into account a content provider's rules on, for example, whether a user will be allowed to make a media content segment, how long the segment would be, what the encoding rate would be. Because there is great variability in the type of rules applicable to each media content item or portion thereof, every piece of content could conceivably have different rules attached to it. Thus, the media content sharing feature should be implemented in accordance with the digital rights management scheme.

It is preferable that the system **600** be implemented in accordance with a digital rights management scheme to guard against the unauthorized exploitation of media content. Absent such a scheme, content providers would be reluctant to entrust the system **600** with their content. Nor would it be prudent for the administrator of the system to entrust that content to end-users. Therefore, the system **600** should provide safeguards media content cannot be easily duplicated or distributed in an unauthorized manner, and ensure that the administrator of the system receives compensation for the use of media content items by end users. Such safeguards would require encrypting the media content items in such a manner that only the users who have paid for the use of the media content items may use them.

A class of software applications known as digital rights management systems has been developed to meet the protection requirements of the content producers. Digital rights management ensures that only authorized users can use a media content item. Furthermore, digital rights management allows for the application of very sophisticated rules for the use of this content. By way of

example, a user may be able to playback a media content item a certain number of times for no fee. Thereafter, the user can playback the media content item additional times for a particular sum of money. Alternatively, a user may purchase the media content item outright for another particular sum. Thus, there is a close relationship between digital rights management, e-commerce and billing. Examples of digital rights management schemes are featured in U.S. Patent No. 6,185,683 and U.S. Patent No. 6,253,193, which are all incorporated herein by reference and assigned to Intertrust Technologies Corp. Intertrust Technologies Corp. has developed digital rights management software, the MetaTrust Utility®, which may be implemented as a digital rights management scheme in the system **600** and the components used in the system **600**. In addition, Microsoft has published a reference, "Digital Rights Management for Microsoft Windows Media Technologies", in 2001. Microsoft's Windows Media Rights Manager® may also be implemented as a digital rights management scheme in the system **600** and the components used in the system **600**. Additional examples of digital rights management schemes are featured in U.S. Patent No. 5,530,235; U.S. Patent No. 5,629,980; U.S. Patent No. 5,634,012; U.S. Patent No. 5,638,443; U.S. Patent No. 6,233,684; and U.S. Patent No. 6,236,971; which are all incorporated herein by reference.

The general trend of the consumer electronics industry is to put data, audio and video capabilities on smaller and smaller portable devices, such as personal media players. Therefore, the second software module has the capability to transfer media files to/from portable devices in accordance with a digital rights

management scheme. The second software module may incorporate well-known libraries for personal media player support. Another way to transfer media content items using the second software module would be to permit users to create CD-ROMs containing media content items in accordance with a digital rights management scheme.

A full variety of content purchase transactions may be accommodated using the system **600**. These transactions include monthly fees for bundles of series ("packages"); purchases of individual series ("a la carte"); purchases of individual episodes ("pay per view" or "PPV"). In addition to managing the how much content the end user receives, digital rights management also manages how long such media content items remain available. For example, a user may have access to a particular media content item in perpetuity, or just provide access to a particular media content item for a predetermined period of time (*e.g.*, a single day).

The system **600** may implement various advertising schemes. By way of example, data, audio and video ads may be placed directly in the media itself, much in the same way that television ads are placed between shows. This form of media advertising is referred to as an "In-Media Ad". In addition, traditional HTML ads (e.g., banners, b-boxes) may be associated with media content items and displayed in an additional window pane of the graphical user interface **700**. These HTML ads could have hotspots which, when clicked using a mouse, would take the user to a special promotional page or a third-party vendor's website. These are called "Third-Pane Ads". Or a full multimedia advertisement could be

delivered as an in-box item, just like a new episode. This form of media advertising is referred to as an "Inbox Ad".

The system **600** may implement a traffic monitoring system that can accurately manage this diverse advertising environment. Aggregate behavioral statistics collected from users may be used to precisely target ads based on a host of parameters such as content item; geographic area; user age; user gender; and user income. Thus users may be exposed to different advertising based on their behavioral statistics. By way of example, if a user regularly watches a program called "Bass Fishing Today", that user may be interested in purchasing some gear for a future fishing trip. There are several ways to take advantage of this commercial opportunity. By way of example, a simple affiliation relationship may exist where a button is provided in an additional windowpane of the graphical user interface **700**. By clicking on the button using a mouse, a user may be transferred to a third-party vendor's web site. Alternatively, system **700** may maintain inventory and manages its own e-commerce product clearinghouse.

Preferably, a user will never have to re-type user information to make a purchase from the system. Thus, user data could be exchanged with third-party vendors to automatically set up an account for the user. Alternatively, the purchased merchandise may be billed to the user's system account.

In addition, each monthly subscription bill sent to a user could be used to enable micro-payments. Credit card companies charge two fees for every purchase: a fixed per-transaction fee and then a percentage of the transaction amount. The fixed fee means that purchases under, for example, ten dollars are

pretty expensive. Accordingly, aggregate purchases could be placed on the user's monthly subscription bill and then a fee would only have to be paid to a credit card company based on the larger amount.

If a user has chosen to subscribe to a specific media content item or type of media content item, there may be other subscribers who enjoy the same media content items. The system **600** may provide vehicles for interaction amongst users subscribing to similar media content items. Three conventional techniques for providing such a vehicle include threaded messaging, instant messaging and chat rooms. Threaded messaging, also referred to as bulletin boards, is a virtual location where users converse asynchronously via e-mail type messages. Typically, a user starts a topic, or "thread" and people respond to that message, or respond to the responses thereof. Threaded messaging is useful because the messages accumulate, allowing you to read the entire discussion on a particular topic. Instant messaging is text-based user-to-user communication, which occurs in real-time and does not leave behind a body of messages like threaded messaging. Instant messaging is an extremely popular way for users to communicate over the network (*e.g.*, Internet). Chat rooms are like instant messaging, but it permits group discussions to occur as messages are sent to all users in the chat room. The system **600** could also allow users to rate and/or recommend content items to others.

The operation of system **600** will now be discussed. A user opens the graphical user interface **700** of the first software playback module **660** on a display of the client computer. The user clicks on the Inbox folder **710** using the

mouse associated with the client computer and finds at least five new media content item have been delivered. The adaptive download module on the server computer system has determined that while the user initially registered for a first media content item (or type of media content item), the user does not actually playback the first media content item often. Thus, only a segment of the first media content item has been downloaded.

The user also subscribes to a particular programming series and one of the episodes in that series is included as a second media content item in the Inbox folder **710**. During the playback of the second media content item, there is a review of five different products, as well as five links to the vendors of those products that are displayed in a window pane of the graphical user interface **700**. After hearing the review, the user clicks on one of the links using a mouse and goes to that vendor's website to make a purchase.

A third media content item is included in the Inbox folder **710** and the user decides to watch the show by clicking on that media content item. The third media content item is an episode of a series featuring a comedian doing a stand-up routine. Using the media content sharing feature, the user makes a segment of the media content item, which features the best joke of all and sends it to another user via e-mail. She also places a media content marker at that point so that the user can return to it later. The user may also go to a bulletin board for the third media content item and post the user's thoughts about the comedian. The user may also receive an instant message from another user regarding the third media content item.

A fourth media content item containing a news audio program is included in the Inbox folder **710**. The user decides to listen to the latest installment of the news audio program to which the user subscribes. The sponsor for this content item knows from the statistical user information gathered for that user, that the user is a woman between 35-50 years of age and making between $25,000 to $35,000 a year income. The dynamic advertisement insertion feature inserts the advertisement that will most likely appeal to Jane. In accordance with a digital rights management scheme, the user also has a CD-ROM burned containing the fourth media content item using a CD drive associated with the client computer. Thus, the fourth media content item is available playback.

A fifth media content item is also included in the Inbox folder **710**. It's a pay per view program, which features an interview with an entertainment personality and only costs a small fee. Although the small fee would be added to Jane's regular monthly bill if the user consumes or plays back the fifth media content item, the user is doubtful about the value of the program. Accordingly, the user accesses the server computer's website and checks out the comments and user ratings section of the website to see what other users have thought about the fifth media content item. In addition, the user previews a portion of the fifth media content.

COMPUTER PROGRAM LISTING

```
#! /usr/local/bin/bash
if [ $1 ]
then
      echo Dumping everything from the database to $1
      pg_dumpall > $1
      echo Dump completed.
else
      echo "Usage: db_dump_all.sh <filename>"
fi
```

```
#! /usr/local/bin/bash
if [ $1 ] && [ $2 ]
then
      echo Creating $1 database from sql file $2
      createdb -U pgsql $1
      psql -U postgres -d $1 < $2
      echo Creation completed.
else
      echo "Usage: db_create.sh <database> <sql file>"
fi
```

```
#! /usr/local/bin/bash
if [ $1 ]
then
      echo Dumping everything from the database to $1
      pg_dumpall > $1
      echo Dump completed.
else
      echo "Usage: db_dump_all.sh <filename>"
fi
```

```
#! /usr/local/bin/bash
if [ $1 ]
then
      echo Dumping the database data to $1
      echo "->Enter pushcast for the username and password."
      pg_dump -a -u pushcast > $1
      echo Dump completed.
else
      echo "Usage: db_dump_data.sh <filename>"
fi
```

```
#! /usr/local/bin/bash
if [ $1 ]
then
      echo Dumping the database schema to $1
      echo "->Enter pushcast for the username and password."
      pg_dump -s -u pushcast > $1
      echo Dump completed.
else
      echo "Usage: db_dump.sh <filename>"
fi
```

```
--
-- Selected TOC Entries:
--
\connect - pushcast
--
-- TOC Entry ID 2 (OID 20803)
--
-- Name: pc_user_userid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_user_userid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 3 (OID 20822)
--
-- Name: pc_player_playerid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_player_playerid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 4 (OID 20841)
--
-- Name: pc_delivery_deliveryid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_delivery_deliveryid_seq" start 1 increment 1
maxvalue 2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 5 (OID 20860)
--
-- Name: pc_listen_listenid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_listen_listenid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 6 (OID 20898)
--
-- Name: pc_episode_episodeid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_episode_episodeid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 7 (OID 20917)
--
-- Name: pc_segment_segmentid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_segment_segmentid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 8 (OID 20936)
--
```

```
-- Name: pc_content_contentid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_content_contentid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 9 (OID 20955)
--
-- Name: pc_blurb_blurbid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_blurb_blurbid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 10 (OID 20974)
--
-- Name: pc_session_sessionid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_session_sessionid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 11 (OID 20993)
--
-- Name: pc_promo_email_promoid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_promo_email_promoid_seq" start 1 increment 1
maxvalue 2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 12 (OID 21012)
--
-- Name: pc_package_packageid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_package_packageid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 13 (OID 21031)
--
-- Name: pc_package_en_package_entry_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_package_en_package_entry_seq" start 1 increment 1
maxvalue 2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 15 (OID 21050)
--
-- Name: pc_user Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_user" (
      "userid" integer DEFAULT nextval('pc_user_userid_seq'::text) NOT
NULL,
      "loginname" character varying(15),
```

```
        "pwd" character varying(15) NOT NULL,
        "zip" character(5),
        "lastlogin" timestamp with time zone DEFAULT "timestamp"(('2001-
07-12 13:54:22-05'::"timestamp" - '7 days'::"interval")),
        "email" character varying(255),
        "sex" character(1),
        "income" character varying(255),
        "born" date,
        "bfname" character varying(255),
        "blname" character varying(255),
        "baddl1" character varying(255),
        "baddl2" character varying(255),
        "bcity" character varying(255),
        "bstate" character(2),
        "bzip" character(5),
        "ccnum" character(16),
        "cctype" character varying(255),
        "expdate" character(5),
        "seccode" character(3),
        "sfname" character varying(255),
        "slname" character varying(255),
        "saddl1" character varying(255),
        "saddl2" character varying(255),
        "scity" character varying(255),
        "sstate" character(2),
        "szip" character(5),
        Constraint "pc_user_pkey" Primary Key ("userid")
);

--
-- TOC Entry ID 16 (OID 21106)
--
-- Name: pc_player Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_player" (
        "playerid" integer DEFAULT nextval('pc_player_playerid_seq'::text)
NOT NULL,
        "userid" integer NOT NULL,
        "datecreated" timestamp with time zone DEFAULT
"timestamp"('now'::text),
        Constraint "pc_player_pkey" Primary Key ("playerid")
);

--
-- TOC Entry ID 17 (OID 21123)
--
-- Name: pc_delivery Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_delivery" (
        "deliveryid" integer DEFAULT nextval
('pc_delivery_deliveryid_seq'::text) NOT NULL,
        "playerid" integer,
        "segmentid" integer,
        "contentid" integer,
        "delivered" timestamp with time zone DEFAULT
"timestamp"('now'::text),
        Constraint "pc_delivery_pkey" Primary Key ("deliveryid")
);
```

```
--
-- TOC Entry ID 18 (OID 21142)
--
-- Name: pc_listen Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_listen" (
      "listenid" integer DEFAULT nextval('pc_listen_listenid_seq'::text)
NOT NULL,
      "playerid" integer,
      "userid" integer,
      "contentid" integer,
      "segmentid" integer,
      "starttimestamp" timestamp with time zone,
      "stoptimestamp" timestamp with time zone,
      "startoffset" integer,
      "stopoffset" integer,
      "filename" character varying(255),
      Constraint "pc_listen_pkey" Primary Key ("listenid")
);

--
-- TOC Entry ID 19 (OID 21190)
--
-- Name: pc_episode Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_episode" (
      "episodeid" integer DEFAULT nextval
('pc_episode_episodeid_seq'::text) NOT NULL,
      "seriesid" integer NOT NULL,
      "name" character varying(255),
      "active" boolean DEFAULT 'f'::bool NOT NULL,
      "sequence" integer,
      "timecreated" timestamp with time zone DEFAULT
"timestamp"('now'::text),
      Constraint "pc_episode_pkey" Primary Key ("episodeid")
);

--
-- TOC Entry ID 20 (OID 21211)
--
-- Name: pc_segment Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_segment" (
      "segmentid" integer DEFAULT nextval
('pc_segment_segmentid_seq'::text) NOT NULL,
      "episodeid" integer,
      "contentid" integer,
      "sequence" integer,
      "description" character varying(255),
      Constraint "pc_segment_pkey" Primary Key ("segmentid")
);

--
-- TOC Entry ID 21 (OID 21229)
--
-- Name: pc_content Type: TABLE Owner: pushcast
--
```

```
CREATE TABLE "pc_content" (
      "contentid" integer DEFAULT nextval
('pc_content_contentid_seq'::text) NOT NULL,
      "filename" character varying(255),
      "name" character varying(255),
      "active" boolean DEFAULT 'f'::bool NOT NULL,
      "url" character varying(255),
      "mhtml" character varying(255),
      Constraint "pc_content_pkey" Primary Key ("contentid")
);

--
-- TOC Entry ID 22 (OID 21249)
--
-- Name: pc_blurb Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_blurb" (
      "blurbid" integer DEFAULT nextval('pc_blurb_blurbid_seq'::text)
NOT NULL,
      "contentid" integer,
      "startoffset" integer,
      "stopoffset" integer,
      "description" character varying(255),
      Constraint "pc_blurb_pkey" Primary Key ("blurbid")
);

--
-- TOC Entry ID 23 (OID 21267)
--
-- Name: pc_session Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_session" (
      "sessionid" integer DEFAULT nextval
('pc_session_sessionid_seq'::text) NOT NULL,
      "userid" integer,
      "sessionstamp" timestamp with time zone DEFAULT
"timestamp"('now'::text),
      "active" boolean DEFAULT 'f'::bool NOT NULL,
      Constraint "pc_session_pkey" Primary Key ("sessionid")
);

--
-- TOC Entry ID 24 (OID 21286)
--
-- Name: pc_promo_email Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_promo_email" (
      "promoid" integer DEFAULT nextval
('pc_promo_email_promoid_seq'::text) NOT NULL,
      "headerfrom" character varying(255),
      "headerreplyto" character varying(255),
      "headersubject" character varying(255),
      "body" text,
      Constraint "pc_promo_email_pkey" Primary Key ("promoid")
);

--
-- TOC Entry ID 25 (OID 21319)
```

```
--
-- Name: pc_package Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_package" (
      "packageid" integer DEFAULT nextval
('pc_package_packageid_seq'::text) NOT NULL,
      "package_type" character varying(255),
      "description" character varying(255),
      "icon" character varying(255),
      Constraint "pc_package_pkey" Primary Key ("packageid")
);

--
-- TOC Entry ID 26 (OID 21335)
--
-- Name: pc_subscription_package Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_subscription_package" (
      "packageid" integer NOT NULL,
      "userid" integer NOT NULL,
      "datecreated" timestamp with time zone DEFAULT
"timestamp"('now'::text),
      "dateterminated" timestamp with time zone,
      Constraint "pc_subscription_package_pkey" Primary Key
("packageid", "userid")
);

--
-- TOC Entry ID 27 (OID 21352)
--
-- Name: pc_subscription_series Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_subscription_series" (
      "seriesid" integer NOT NULL,
      "userid" integer NOT NULL,
      "datecreated" timestamp with time zone DEFAULT
"timestamp"('now'::text),
      "datetermintated" timestamp with time zone,
      Constraint "pc_subscription_series_pkey" Primary Key ("seriesid",
"userid")
);

--
-- TOC Entry ID 28 (OID 21369)
--
-- Name: pc_package_entry Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_package_entry" (
      "packageid" integer NOT NULL,
      "seriesid" integer NOT NULL,
      Constraint "pc_package_entry_pkey" Primary Key ("packageid",
"seriesid")
);

--
-- TOC Entry ID 14 (OID 21557)
--
```

```
-- Name: pc_series_seriesid_seq Type: SEQUENCE Owner: pushcast
--

CREATE SEQUENCE "pc_series_seriesid_seq" start 1 increment 1 maxvalue
2147483647 minvalue 1  cache 1 ;

--
-- TOC Entry ID 29 (OID 21576)
--
-- Name: pc_series Type: TABLE Owner: pushcast
--

CREATE TABLE "pc_series" (
      "seriesid" integer DEFAULT nextval
('"pc_series_seriesid_seq"'::text) NOT NULL,
      "active" boolean DEFAULT 'f'::bool NOT NULL,
      "name" character varying(255),
      "description" text,
      "adult" boolean,
      "producer" character varying(255),
      "startdate" date,
      "stopdate" date,
      "frequency" character varying(255),
      "episodelength" character varying(255),
      "language" character varying(255),
      "genre" character varying(255),
      "icon" character varying(255),
      Constraint "pc_series_pkey" Primary Key ("seriesid")
);

--
-- TOC Entry ID 30 (OID 21637)
--
-- Name: pc_subscription Type: VIEW Owner: pushcast
--

CREATE VIEW "pc_subscription" as SELECT pc_series.seriesid,
pc_subscription_package.userid FROM pc_series, pc_subscription_package,
pc_package_entry WHERE ((pc_series.seriesid = pc_package_entry.seriesid)
AND (pc_package_entry.packageid = pc_subscription_package.packageid))
UNION SELECT pc_subscription_series.seriesid,
pc_subscription_series.userid FROM pc_subscription_series;

--
-- TOC Entry ID 31 (OID 21050)
--
-- Name: "pc_user_loginname_key" Type: INDEX Owner: pushcast
--

CREATE UNIQUE INDEX "pc_user_loginname_key" on "pc_user" using btree (
"loginname" "varchar_ops" );

--
-- TOC Entry ID 32 (OID 21190)
--
-- Name: "pc_episode_episodeid_key1" Type: INDEX Owner: pushcast
--

CREATE UNIQUE INDEX "pc_episode_episodeid_key1" on "pc_episode" using
btree ( "episodeid" "int4_ops", "name" "varchar_ops" );
```

```
--
-- TOC Entry ID 33 (OID 21190)
--
-- Name: "pc_episode_episodeid_key2" Type: INDEX Owner: pushcast
--

CREATE UNIQUE INDEX "pc_episode_episodeid_key2" on "pc_episode" using
btree ( "episodeid" "int4_ops", "sequence" "int4_ops" );

--
-- TOC Entry ID 34 (OID 21408)
--
-- Name: "RI_ConstraintTrigger_21407" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_player', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 35 (OID 21410)
--
-- Name: "RI_ConstraintTrigger_21409" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_player', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 36 (OID 21412)
--
-- Name: "RI_ConstraintTrigger_21411" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_listen', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 37 (OID 21414)
--
-- Name: "RI_ConstraintTrigger_21413" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_listen', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 38 (OID 21416)
--
-- Name: "RI_ConstraintTrigger_21415" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
```

```
"RI_FKey_noaction_del" ('<unnamed>', 'pc_session', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 39 (OID 21418)
--
-- Name: "RI_ConstraintTrigger_21417" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_session', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 40 (OID 21420)
--
-- Name: "RI_ConstraintTrigger_21419" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_subscription_package',
'pc_user', 'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 41 (OID 21422)
--
-- Name: "RI_ConstraintTrigger_21421" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_subscription_package',
'pc_user', 'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 42 (OID 21424)
--
-- Name: "RI_ConstraintTrigger_21423" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_subscription_series',
'pc_user', 'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 43 (OID 21426)
--
-- Name: "RI_ConstraintTrigger_21425" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_user"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_subscription_series',
'pc_user', 'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 44 (OID 21428)
--
```

```
-- Name: "RI_ConstraintTrigger_21427" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_player"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_player', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 45 (OID 21430)
--
-- Name: "RI_ConstraintTrigger_21429" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_player"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_delivery', 'pc_player',
'UNSPECIFIED', 'playerid', 'playerid');

--
-- TOC Entry ID 46 (OID 21432)
--
-- Name: "RI_ConstraintTrigger_21431" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_player"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_delivery', 'pc_player',
'UNSPECIFIED', 'playerid', 'playerid');

--
-- TOC Entry ID 47 (OID 21434)
--
-- Name: "RI_ConstraintTrigger_21433" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_delivery"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_delivery', 'pc_player',
'UNSPECIFIED', 'playerid', 'playerid');

--
-- TOC Entry ID 48 (OID 21436)
--
-- Name: "RI_ConstraintTrigger_21435" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_listen"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_listen', 'pc_listen',
'UNSPECIFIED', 'playerid', 'listenid');

--
-- TOC Entry ID 49 (OID 21438)
--
-- Name: "RI_ConstraintTrigger_21437" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_listen"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_listen', 'pc_listen',
```

```
'UNSPECIFIED', 'playerid', 'listenid');

--
-- TOC Entry ID 50 (OID 21440)
--
-- Name: "RI_ConstraintTrigger_21439" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_listen"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_listen', 'pc_listen',
'UNSPECIFIED', 'playerid', 'listenid');

--
-- TOC Entry ID 51 (OID 21442)
--
-- Name: "RI_ConstraintTrigger_21441" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_listen"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_listen', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 52 (OID 21456)
--
-- Name: "RI_ConstraintTrigger_21455" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_episode"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_episode', 'pc_series',
'UNSPECIFIED', 'seriesid', 'seriesid');

--
-- TOC Entry ID 53 (OID 21458)
--
-- Name: "RI_ConstraintTrigger_21457" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_episode"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_segment', 'pc_episode',
'UNSPECIFIED', 'episodeid', 'episodeid');

--
-- TOC Entry ID 54 (OID 21460)
--
-- Name: "RI_ConstraintTrigger_21459" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_episode"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_segment', 'pc_episode',
'UNSPECIFIED', 'episodeid', 'episodeid');

--
-- TOC Entry ID 55 (OID 21462)
--
-- Name: "RI_ConstraintTrigger_21461" Type: TRIGGER Owner: pushcast
```

```
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_segment"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_segment', 'pc_episode',
'UNSPECIFIED', 'episodeid', 'episodeid');

--
-- TOC Entry ID 56 (OID 21464)
--
-- Name: "RI_ConstraintTrigger_21463" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_content"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_blurb', 'pc_content',
'UNSPECIFIED', 'contentid', 'contentid');

--
-- TOC Entry ID 57 (OID 21466)
--
-- Name: "RI_ConstraintTrigger_21465" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_content"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_blurb', 'pc_content',
'UNSPECIFIED', 'contentid', 'contentid');

--
-- TOC Entry ID 58 (OID 21468)
--
-- Name: "RI_ConstraintTrigger_21467" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_blurb"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_blurb', 'pc_content',
'UNSPECIFIED', 'contentid', 'contentid');

--
-- TOC Entry ID 59 (OID 21470)
--
-- Name: "RI_ConstraintTrigger_21469" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_session"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE
PROCEDURE "RI_FKey_check_ins" ('<unnamed>', 'pc_session', 'pc_user',
'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 60 (OID 21472)
--
-- Name: "RI_ConstraintTrigger_21471" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_package"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_package_entry', 'pc_package',
'UNSPECIFIED', 'packageid', 'packageid');
```

```
--
-- TOC Entry ID 61 (OID 21474)
--
-- Name: "RI_ConstraintTrigger_21473" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_package"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_package_entry', 'pc_package',
'UNSPECIFIED', 'packageid', 'packageid');

--
-- TOC Entry ID 62 (OID 21476)
--
-- Name: "RI_ConstraintTrigger_21475" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_package"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_del" ('<unnamed>', 'pc_subscription_package',
'pc_package', 'UNSPECIFIED', 'packageid', 'packageid');

--
-- TOC Entry ID 63 (OID 21478)
--
-- Name: "RI_ConstraintTrigger_21477" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_package"  NOT
DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW EXECUTE PROCEDURE
"RI_FKey_noaction_upd" ('<unnamed>', 'pc_subscription_package',
'pc_package', 'UNSPECIFIED', 'packageid', 'packageid');

--
-- TOC Entry ID 66 (OID 21480)
--
-- Name: "RI_ConstraintTrigger_21479" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_subscription_package"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH
ROW EXECUTE PROCEDURE "RI_FKey_check_ins" ('<unnamed>',
'pc_subscription_package', 'pc_package', 'UNSPECIFIED', 'packageid',
'packageid');

--
-- TOC Entry ID 67 (OID 21482)
--
-- Name: "RI_ConstraintTrigger_21481" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_subscription_package"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH
ROW EXECUTE PROCEDURE "RI_FKey_check_ins" ('<unnamed>',
'pc_subscription_package', 'pc_user', 'UNSPECIFIED', 'userid',
'userid');

--
-- TOC Entry ID 68 (OID 21484)
--
```

```
-- Name: "RI_ConstraintTrigger_21483" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_subscription_series"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH
ROW EXECUTE PROCEDURE "RI_FKey_check_ins" ('<unnamed>',
'pc_subscription_series', 'pc_series', 'UNSPECIFIED', 'seriesid',
'seriesid');

--
-- TOC Entry ID 69 (OID 21486)
--
-- Name: "RI_ConstraintTrigger_21485" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_subscription_series"  NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH
ROW EXECUTE PROCEDURE "RI_FKey_check_ins" ('<unnamed>',
'pc_subscription_series', 'pc_user', 'UNSPECIFIED', 'userid', 'userid');

--
-- TOC Entry ID 70 (OID 21488)
--
-- Name: "RI_ConstraintTrigger_21487" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER INSERT OR UPDATE ON
"pc_package_entry"  FROM "pc_package" NOT DEFERRABLE INITIALLY IMMEDIATE
FOR EACH ROW EXECUTE PROCEDURE "RI_FKey_check_ins" ('<unnamed>',
'pc_package_entry', 'pc_package', 'UNSPECIFIED', 'packageid',
'packageid');

--
-- TOC Entry ID 64 (OID 21490)
--
-- Name: "RI_ConstraintTrigger_21489" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER DELETE ON "pc_package"  FROM
"pc_package_entry" NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW
EXECUTE PROCEDURE "RI_FKey_noaction_del" ('<unnamed>',
'pc_package_entry', 'pc_package', 'UNSPECIFIED', 'packageid',
'packageid');

--
-- TOC Entry ID 65 (OID 21492)
--
-- Name: "RI_ConstraintTrigger_21491" Type: TRIGGER Owner: pushcast
--

CREATE CONSTRAINT TRIGGER "<unnamed>" AFTER UPDATE ON "pc_package"  FROM
"pc_package_entry" NOT DEFERRABLE INITIALLY IMMEDIATE FOR EACH ROW
EXECUTE PROCEDURE "RI_FKey_noaction_upd" ('<unnamed>',
'pc_package_entry', 'pc_package', 'UNSPECIFIED', 'packageid',
'packageid');
```

```
#include "stdafx.h"
#include "player.h"
#include "DisplayView.h"
#include "PlayerDoc.h"
#include "Windowsx.h" // for GET_X_LPARAM
#include <libxml/tree.h>
#include <algorithm>
#include "utils.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CDisplayView, CReportView)

BEGIN_MESSAGE_MAP(CDisplayView, CReportView)
      //{{AFX_MSG_MAP(CDisplayView)
      ON_NOTIFY(RVN_ITEMRCLICK, 0, OnRclick)
      ON_NOTIFY(RVN_ITEMCLICK, 0, OnClick)
      ON_COMMAND(ID_DISPLAY_DELETE, OnDeleteMedia)
      ON_COMMAND(ID_DISPLAY_PLAY, OnPlayMedia)
      ON_NOTIFY(RVN_ITEMDBCLICK, 0, OnDblclk)
      ON_NOTIFY(RVN_HEADERCLICK, 0, OnColumnClick)
      ON_NOTIFY(RVN_ITEMCALLBACK, 0, OnRvnItemCallback)
      ON_NOTIFY(RVN_SELECTIONCHANGED, 0, OnRvnSelectionChanged)
      ON_NOTIFY(RVN_KEYDOWN, 0, OnNmReturn) // 4294965237
      ON_COMMAND(ID_VIEW_SHOWVGRID, OnViewShowVGrid)
      ON_UPDATE_COMMAND_UI(ID_VIEW_SHOWVGRID, OnUpdateViewShowVGrid)
      ON_COMMAND(ID_VIEW_SHOWHGRID, OnViewShowHGrid)
      ON_UPDATE_COMMAND_UI(ID_VIEW_SHOWHGRID, OnUpdateViewShowHGrid)
      ON_COMMAND(ID_VIEW_SHOWHGRIDEX, OnViewShowHGridEx)
      ON_UPDATE_COMMAND_UI(ID_VIEW_SHOWHGRIDEX, OnUpdateViewShowHGridEx)
      ON_COMMAND(ID_VIEW_ALTERNATECOLORS, OnViewAlternateColors)
      ON_UPDATE_COMMAND_UI(ID_VIEW_ALTERNATECOLORS,
OnUpdateViewAlternateColors)
      ON_COMMAND(ID_DELETE, OnDeleteMedia)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CDisplayView::CDisplayView() {
      m_iSelected = RVI_INVALID;
}

////////////////////////////////////////////////////////////////////////
/////
// CDisplayView drawing

void CDisplayView::OnDraw(CDC* pDC)
{
      CDocument* pDoc = GetDocument();
      // TODO: add draw code here
}

////////////////////////////////////////////////////////////////////////
/////
// CDisplayView diagnostics

#ifdef _DEBUG
void CDisplayView::AssertValid() const
```

```
{
	CReportView::AssertValid();
}

void CDisplayView::Dump(CDumpContext& dc) const
{
	CReportView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
/////
// CDisplayView message handlers

void CDisplayView::OnInitialUpdate()
{
	CReportView::OnInitialUpdate();


	/*------------------------------------------------
	-BEGIN- Add columns to the list view.
	------------------------------------------------*/
	RVSUBITEM rvs;

	rvs.lpszText = _T("Show");
	rvs.iWidth = 170;
	GetReportCtrl().DefineSubItem(0, &rvs);
	GetReportCtrl().ActivateSubItem(0, 0);

	/*rvs.lpszText = _T("Episode");
	rvs.iWidth = 50;
	GetReportCtrl().DefineSubItem(1, &rvs);
	GetReportCtrl().ActivateSubItem(1, 1);*/

	rvs.lpszText = _T("Description");
	rvs.iWidth = 240;
	GetReportCtrl().DefineSubItem(1, &rvs);
	GetReportCtrl().ActivateSubItem(1, 1);

	rvs.lpszText = _T("Received");
	rvs.iWidth = 140;
	GetReportCtrl().DefineSubItem(2, &rvs);
	GetReportCtrl().ActivateSubItem(2, 2);


	/* Published used to be here */

	rvs.lpszText = _T("Days To Trash");
	rvs.iWidth = 100;
	GetReportCtrl().DefineSubItem(3, &rvs);
	GetReportCtrl().ActivateSubItem(3, 3);
	/*------------------------------------------------
	-END- Add columns to the list view.
	------------------------------------------------*/

	/*------------------------------------------------
	-BEGIN- Use Transparency
	------------------------------------------------*/
	/*WCHAR   wszWallpaper [MAX_PATH];
	CString strPath;
```

```
    HRESULT hr;
    IActiveDesktop* pIAD;

  // 1. Initialize the COM library (make Windows load the DLLs).
Normally you would call
  // this in your InitInstance() or other startup code.  In MFC apps,
use AfxOleInit() instead.
  CoInitialize ( NULL );

  // 2. Create a COM object, using the Active Desktop coclass provided
by the shell.
  // The 4th parameter tells COM what interface we want
(IActiveDesktop).
  hr = CoCreateInstance ( CLSID_ActiveDesktop,
                          NULL,
                          CLSCTX_INPROC_SERVER,
                          IID_IActiveDesktop,
                          (void**) &pIAD );
  if ( SUCCEEDED(hr) )
     {
     // 3. If the COM object was created, call its GetWallpaper()
method.
     hr = pIAD->GetWallpaper ( wszWallpaper, MAX_PATH, 0 );

     if ( SUCCEEDED(hr) )
        {
        // 4. If GetWallpaper() succeeded, print the filename it
returned.
        // Note that I'm using wcout to display the Unicode string
wszWallpaper.  wcout is
        // the Unicode equivalent of cout.
            CString sWallpaper = wszWallpaper;
            GetReportCtrl().SetBkImage(sWallpaper);
        }
     else
        {
        }

     // 5. Release the interface.
     pIAD->Release();
     }
  else
     {
     }

  // 6. Uninit the COM library.  In MFC apps, this is not necessary
since MFC does it for us.
  CoUninitialize();
    *//*-----------------------------------------------
    -END- Use Transparency
    -----------------------------------------------*/

    m_oleDropTarget.Register(this);
}

BOOL CDisplayView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.style |= RVS_OWNERDATA;

    GetReportCtrl().InsertColor(0, 0x00C0D8C0);
    GetReportCtrl().InsertColor(1, ::GetSysColor(COLOR_GRAYTEXT));
```

```
	GetReportCtrl().InsertColor(2, 0x00D0C0C0);
	GetReportCtrl().InsertColor(3, 0x00804000);

	// We need read too
	GetReportCtrl().InsertColor(4, 0x000000FF);

	return CReportView::PreCreateWindow(cs);
}

void CDisplayView::OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint)
{
	CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
	if(pDoc->GetCurSelectedFolder() != NULL) {
		GetReportCtrl().DeleteAllItems();
		AddMediaFromXML(pDoc->GetCurSelectedFolder());
	}
}

void CDisplayView::AddMediaFromXML(xmlNodePtr node)
{
	// Remember the working folder node
	currentFolderNode = node;

	CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
	// Setup the proper columns...ie, the inbox and trash need Days to
Purge
	SetupColumns();

	// Purge any item
//	pDoc->purgeItems(node);

	// Clear current item data from previous folder.
	currentItemData.clear();

	// Set item data for current folder.
	for(xmlNodePtr item = node->children; item != NULL; item = item->
next) {
		if(strcmp(reinterpret_cast<const char *>(item->name),
"Media") == 0) {
			currentItemData.push_back(item);
		}
	}

	// Configure the list widget.
	GetReportCtrl().FlushCache();
	GetReportCtrl().SetItemCount(currentItemData.size());
}

void CDisplayView::OnRclick(NMHDR* pNMHDR, LRESULT* pResult)
{
	CReportCtrl & lc = GetReportCtrl();

	/* Get the mouse cursor position */
	DWORD dwPos = GetMessagePos();

	/* Convert the co-ords into a CPoint structure */
	CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
	spt = pt;

	/* Convert to screen co-ords for hittesting */
```

```
        lc.ScreenToClient( &spt );
        // Set rightClickPoint incase someone needs it in the future
        // for hittesting.

        rightClickPoint = spt;

        // Add Popup menu
        CMenu menu;
        VERIFY(menu.LoadMenu(IDR_DISPLAY_MENU));
        CMenu* pPopup = menu.GetSubMenu(0);
        ASSERT(pPopup != NULL);

        pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, pt.x,
pt.y, AfxGetMainWnd());

        *pResult = 0;
}

// Purpose: Delete the selected item.  If item is in the trash and not
local,
//                delete it from the drive, otherwise remove the
xmlnode.
void CDisplayView::OnDeleteMedia()
{
        // Get the Doc and List Control.
        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
        CReportCtrl &lc = GetReportCtrl();

        // Get the position of the first selected item.
        int pos = lc.GetFirstSelectedItem();

        // If something is selected, do the following.
        if (pos) {
                while (pos) {
                        // Get the selected item.
                        int nItem = lc.GetNextSelectedItem(pos);
                        TRACE1("Item %d was selected!\n", nItem);
                        //xmlNodePtr node = (xmlNodePtr)lc.GetItemData(nItem);
                        xmlNodePtr node = currentItemData[pos];

                        // Check if item is in the trash.
                        if(pDoc->InTrash()) {
                                // File is in trash so check if not local.
                                xmlChar *show = xmlGetProp(node,
                                        reinterpret_cast<const unsigned char
 *> ("SHOW"));
                                string strShow = reinterpret_cast<const char *>
 (show);
                                if(!(strShow == "Local")) {
                                        //File was not local so delete from drive.
                                        xmlChar *url = xmlGetProp(node,
                                                reinterpret_cast<const
 unsigned char *> ("URL"));
                                        string strURL = reinterpret_cast<const
 char *>(url);
                                        if (!DeleteFile(strURL.c_str()))
                                                MessageBox("File not
 deleted.",NULL,MB_OK);
                                }

                        // File was not in trash so move it there.
```

```
                    } else {
                        pDoc->MoveNodeToFolderNode(node, NULL);

                    }

                    // Delete node from list and xml.
                    //lc.DeleteItem(nItem);
                    currentItemData.erase(currentItemData.begin()+pos);
                    lc.FlushCache();
                    lc.SetItemCount(currentItemData.size());
                    lc.Invalidate();
                    xmlUnlinkNode(node);

                    // Get the next selected item if there is one.
                    //pos = lc.GetFirstSelectedItem();
                    pos = false;
                }
        // Otherwise no items were selected.
        } else {
            TRACE0("No items were selected!\n");
        }
}

// Opens the media that is double clicked by sending a message to the
MainFrame.
void CDisplayView::OnDblclk(NMHDR* pNMHDR, LRESULT* pResult)
{
    ::PostMessage(GetParentFrame()->m_hWnd,WM_USER_PLAY,(WPARAM)0,
(LPARAM)0);
    *pResult = 0;
}

bool CDisplayView::SetupColumns()
{
    if(GetReportCtrl().GetActiveSubItemCount()) {
    CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
    CHeaderCtrl* headCtrl = GetReportCtrl().GetHeaderCtrl();
    /* Should trash purge itself?..Should they Know? */
    //if(pDoc->InInbox() || pDoc->InTrash()) {
    if(pDoc->InInbox()) {
        if(!GetReportCtrl().IsActiveSubItem(3)) {
            GetReportCtrl().ActivateSubItem(3, 3);
            return true;
        }
        else {
            return true;
        }
    }

    else {
        if(GetReportCtrl().IsActiveSubItem(3)) {
            GetReportCtrl().DeactivateSubItem(3);
            return true;
        }
        else {
            return true;
        }
    }
    }

    return true;
```

```
}

void CDisplayView::OnClick(NMHDR* pNMHDR, LRESULT* pResult)
{
      /*-----------------------------------------------------
      -BEGIN- Find the Item Clicked on
      -----------------------------------------------------*/
      /* Get the mouse cursor position */
      DWORD dwPos = GetMessagePos();

      /* Convert the co-ords into a CPoint structure */
      CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
      spt = pt;

      /* Convert to screen co-ords for hittesting */
      GetReportCtrl().ScreenToClient( &spt );

      RVHITTESTINFO hti;
      hti.point = spt;
      int selectedItem = GetReportCtrl().HitTest(&hti);
      if(selectedItem < 0) return;
      /*-----------------------------------------------------
      -END- Find the Item Clicked on
      -----------------------------------------------------*/

      /*-----------------------------------------------------
      -BEGIN- drag-and-drop
      -----------------------------------------------------*/
      typedef struct { xmlNodePtr x; } S;
      HANDLE hData = ::GlobalAlloc (GMEM_MOVEABLE, sizeof (S));
      S* s = (S*) ::GlobalLock (hData);
      s->x = xmlCopyNode(currentItemData[selectedItem],1);
      ::GlobalUnlock (hData);

      COleDataSource ods;
      UINT nFormat = ((CPlayerApp*) AfxGetApp ())->GetClipboardFormat
();
      ods.CacheGlobalData (nFormat, hData);

      int nOldSel = selectedItem;
      DROPEFFECT de = ods.DoDragDrop (DROPEFFECT_COPY |
DROPEFFECT_MOVE);

      if (de == DROPEFFECT_MOVE) {
            xmlUnlinkNode(currentItemData[selectedItem]);
            xmlFreeNode(currentItemData[selectedItem]);
            currentItemData.erase(currentItemData.begin()+selectedItem);
            GetReportCtrl().SetItemCount(currentItemData.size());
            CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>
(GetDocument());
            pDoc->UpdateAllViews(NULL);
      }

      xmlFreeNode(s->x);
      ::GlobalFree(hData);
      /*-----------------------------------------------------
      -END- drag-and-drop
      -----------------------------------------------------*/
}

void CDisplayView::OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult)
```

```
{
      LPNMRVHEADER lpnmrv = (LPNMRVHEADER)pNMHDR;

      switch(lpnmrv->iSubItem) {
      case 0: { // Show
            std::sort(currentItemData.begin(), currentItemData.end(),
SShowSort());
            break;
                  }
      case 1: { // Episode
            std::sort(currentItemData.begin(), currentItemData.end(),
SEpisodeSort());
            break;
                  }
      case 2: { // Description
            std::sort(currentItemData.begin(), currentItemData.end(),
SDescriptionSort());
            break;
                  }
      case 3: { // Received
            std::sort(currentItemData.begin(), currentItemData.end(),
SReceivedSort());
            break;
                  }
      case 4: { // Published
            std::sort(currentItemData.begin(), currentItemData.end(),
SPublishedSort());
            break;
                  }
      case 5: { // Days To Purge
            std::sort(currentItemData.begin(), currentItemData.end(),
SDaysToPurgeSort());
            break;
                  }
      }

      GetReportCtrl().FlushCache();
      GetReportCtrl().Invalidate();

      *pResult = 0;
}

void CDisplayView::OnRvnItemCallback(NMHDR *pNMHDR, LRESULT *pResult)
{
      /* Based on Figure 10-9 page 602, "Programming Windows with MFC
         2nd Ed." by Jeff Prosise */

      LPNMRVITEMCALLBACK lpnmrvic = (LPNMRVITEMCALLBACK)pNMHDR;

      int daystopurge = 1;
      xmlNodePtr item = currentItemData[lpnmrvic->item.iItem];
      if ((item == NULL) || (lpnmrvic->item.iItem < 0)) return;

      switch(lpnmrvic->item.iSubItem) {
      case 0: { // Show
            xmlChar *show = xmlGetProp(item,
                  reinterpret_cast<const unsigned char *>("SHOW"));
            string strShow = reinterpret_cast<const char *>(show);

            _tcscpy(lpnmrvic->item.lpszText, strShow.c_str());
```

```
            break;
                  }
      /*case 1: { // Episode
            xmlChar *episode = xmlGetProp(item,
                  reinterpret_cast<const unsigned char *>("EPISODE"));
            string strEpisode = reinterpret_cast<const char *>(episode);

            _tcscpy(lpnmrvic->item.lpszText, strEpisode.c_str());

            break;
                  }*/
      case 1: { // Description
            xmlChar *desc = xmlGetProp(item,
                  reinterpret_cast<const unsigned char *>("DESC"));
            string strDesc = reinterpret_cast<const char *>(desc);

            _tcscpy(lpnmrvic->item.lpszText, strDesc.c_str());

            break;
                  }
      case 2: { // Received
            xmlChar *received = xmlGetProp(item,
                  reinterpret_cast<const unsigned char *>("RECEIVED"));
            string strReceived = reinterpret_cast<const char *>
(received);

            _tcscpy(lpnmrvic->item.lpszText, strReceived.c_str());

            break;
                  }

      case 3: { // Days to Purge
            CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>
(GetDocument());

            daystopurge = 1;

            xmlChar *received = xmlGetProp(item,
                  reinterpret_cast<const unsigned char *>("RECEIVED"));
            string strReceived = reinterpret_cast<const char *>
(received);

            if((pDoc->InInbox() || pDoc->InTrash()) && (strReceived !=
"-")) {
                  char strDays[255];

                  CTime recvCTime = CUtils::BuildCTime(strReceived.c_str
());
                  CTime curCTime = CTime::GetCurrentTime();
                  CTimeSpan elapsedTime = curCTime - recvCTime;
                  daystopurge = pDoc->GetDaysToPurgeInbox() -
elapsedTime.GetDays();
                  sprintf(strDays, "%d", daystopurge);
                  _tcscpy(lpnmrvic->item.lpszText, strDays);
            }

            break;
                  }
      } // end switch

      /* ----------------------------------------------
```

```
        -BEGIN- Mark unheard as bold
        ----------------------------------------------*/
        xmlChar *listened = xmlGetProp(item,
              reinterpret_cast<const unsigned char *>("LISTENED"));
        string strListened = "0";
        if(listened != NULL)
              strListened = reinterpret_cast<const char *>(listened);
        if(strListened == "0") {
              lpnmrvic->item.nMask |= RVIM_STATE;
              lpnmrvic->item.nState |= RVIS_BOLD;
        }
        /* ----------------------------------------------
        -END- Mark unheard as bold
        ----------------------------------------------*/

        /* ----------------------------------------------
        -BEGIN- Mark read if soon to be purged
        ----------------------------------------------*/
              // Setup RED as a color we can set items too.

        if(daystopurge <= 0) { // draw as red
              lpnmrvic->item.iTextColor = 4;
              lpnmrvic->item.nMask |= RVIM_TEXTCOLOR;
              _tcscpy(lpnmrvic->item.lpszText,"Will Be Purged");
        }


        /* ----------------------------------------------
        -END- Mark read if soon to be purged
        ----------------------------------------------*/


        if(lpnmrvic->item.iItem == m_iSelected)
        {
              lpnmrvic->item.nMask |= RVIM_STATE;
              lpnmrvic->item.nState |= RVIS_SELECTED;
        }

        pResult = FALSE;
}

void CDisplayView::OnRvnSelectionChanged(NMHDR* pNMHDR, LRESULT*
pResult)
{
        LPNMREPORTVIEW lpnmrv = (LPNMREPORTVIEW)pNMHDR;

        if(lpnmrv->nState&RVIS_SELECTED) {
              // Get the data for the current item.
              xmlNodePtr item = currentItemData[lpnmrv->iItem];
              if (item != NULL) {
                    // Set the current item in the document.
                    CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>
(GetDocument());
                    pDoc->SetCurSelectedItem(item);
                    ::PostMessage(this->GetParentFrame()->
m_hWnd,WM_USER_LOADURL,(WPARAM)0, (LPARAM)0);
              }

              m_iSelected = lpnmrv->iItem;
        }
```

```
        pResult = FALSE;
    }

    bool SShowSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
    {
        xmlChar *show1 = xmlGetProp(item1,
              reinterpret_cast<const unsigned char *>("SHOW"));
        string strShow1 = reinterpret_cast<const char *>(show1);

        xmlChar *show2 = xmlGetProp(item2,
              reinterpret_cast<const unsigned char *>("SHOW"));
        string strShow2 = reinterpret_cast<const char *>(show2);

        return (CUtils::cmp_nocase(strShow1,strShow2) < 0) ? true : false;
    }

    bool SEpisodeSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
    {
        xmlChar *episode1 = xmlGetProp(item1,
              reinterpret_cast<const unsigned char *>("EPISODE"));
        string strEpisode1 = reinterpret_cast<const char *>(episode1);

        xmlChar *episode2 = xmlGetProp(item2,
              reinterpret_cast<const unsigned char *>("EPISODE"));
        string strEpisode2 = reinterpret_cast<const char *>(episode2);

        return (CUtils::cmp_nocase(strEpisode1,strEpisode2) < 0) ? true :
    false;
    }

    bool SDescriptionSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
    {
        xmlChar *desc1 = xmlGetProp(item1,
              reinterpret_cast<const unsigned char *>("DESC"));
        string strDesc1 = reinterpret_cast<const char *>(desc1);

        xmlChar *desc2 = xmlGetProp(item2,
              reinterpret_cast<const unsigned char *>("DESC"));
        string strDesc2 = reinterpret_cast<const char *>(desc2);

        return (CUtils::cmp_nocase(strDesc1,strDesc2) < 0) ? true : false;
    }

    bool SReceivedSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
    {
        xmlChar *received1 = xmlGetProp(item1,
              reinterpret_cast<const unsigned char *>("RECEIVED"));
        string strReceived1 = reinterpret_cast<const char *>(received1);

        xmlChar *received2 = xmlGetProp(item2,
              reinterpret_cast<const unsigned char *>("RECEIVED"));
        string strReceived2 = reinterpret_cast<const char *>(received2);

        if (strReceived1 == "-" && strReceived2 == "-") {
              return false;
        }
        if (strReceived1 == "-" && strReceived2 != "-") {
              return true;
        }
        if (strReceived1 != "-" && strReceived2 == "-") {
              return false;
```

```
        }

        CTime curCTime = CTime::GetCurrentTime();
        CTime recvCTime1 = CUtils::BuildCTime(strReceived1.c_str());
        CTime recvCTime2 = CUtils::BuildCTime(strReceived2.c_str());
        CTimeSpan ts = recvCTime1 - recvCTime2;

        return (ts.GetTotalSeconds() < 0) ? true : false;
}

bool SPublishedSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
{
        xmlChar *published1 = xmlGetProp(item1,
                reinterpret_cast<const unsigned char *>("PUBLISHED"));
        string strPublished1 = reinterpret_cast<const char *>(published1);

        xmlChar *published2 = xmlGetProp(item2,
                reinterpret_cast<const unsigned char *>("PUBLISHED"));
        string strPublished2 = reinterpret_cast<const char *>(published2);

        if (strPublished1 == "-" && strPublished2 == "-") {
                return false;
        }
        if (strPublished1 == "-" && strPublished2 != "-") {
                return true;
        }
        if (strPublished1 != "-" && strPublished2 == "-") {
                return false;
        }

        CTime pubCTime1 = CUtils::BuildCTime(strPublished1.c_str());
        CTime pubCTime2 = CUtils::BuildCTime(strPublished2.c_str());
        CTimeSpan ts = pubCTime1 - pubCTime2;

        return (ts.GetTotalSeconds() < 0) ? true : false;
}

bool SDaysToPurgeSort::operator()(xmlNodePtr& item1, xmlNodePtr& item2)
{
        xmlChar *received1 = xmlGetProp(item1,
                reinterpret_cast<const unsigned char *>("RECEIVED"));
        string strReceived1 = reinterpret_cast<const char *>(received1);

        xmlChar *received2 = xmlGetProp(item2,
                reinterpret_cast<const unsigned char *>("RECEIVED"));
        string strReceived2 = reinterpret_cast<const char *>(received2);

        if (strReceived1 == "-" && strReceived2 == "-") {
                return false;
        }
        if (strReceived1 == "-" && strReceived2 != "-") {
                return true;
        }
        if (strReceived1 != "-" && strReceived2 == "-") {
                return false;
        }

        CTime curCTime = CTime::GetCurrentTime();
        CTime recvCTime1 = CUtils::BuildCTime(strReceived1.c_str());
        CTime recvCTime2 = CUtils::BuildCTime(strReceived2.c_str());
        CTimeSpan elapsedTime1 = curCTime - recvCTime1;
```

```
        CTimeSpan elapsedTime2 = curCTime - recvCTime2;
        int daystopurge1 =
              elapsedTime1.GetDays();
        int daystopurge2 =
              elapsedTime2.GetDays();

        return ((daystopurge1 - daystopurge2) < 0) ? true : false;
}

void CDisplayView::OnViewShowVGrid()
{
        CReportCtrl& rc = GetReportCtrl();
        if(rc.GetStyle()&RVS_SHOWVGRID)
              rc.ModifyStyle(RVS_SHOWVGRID, 0);
        else
              rc.ModifyStyle(0, RVS_SHOWVGRID);
}

void CDisplayView::OnUpdateViewShowVGrid(CCmdUI* pCmdUI)
{
        CReportCtrl& rc = GetReportCtrl();
        pCmdUI->SetCheck(rc.GetStyle()&RVS_SHOWVGRID);
}

void CDisplayView::OnViewShowHGrid()
{
        CReportCtrl& rc = GetReportCtrl();
        if(rc.GetStyle()&RVS_SHOWHGRID)
              rc.ModifyStyle(RVS_SHOWHGRID, 0);
        else
              rc.ModifyStyle(0, RVS_SHOWHGRID);
}

void CDisplayView::OnUpdateViewShowHGrid(CCmdUI* pCmdUI)
{
        CReportCtrl& rc = GetReportCtrl();
        pCmdUI->SetCheck(rc.GetStyle()&RVS_SHOWHGRID);
}

void CDisplayView::OnViewShowHGridEx()
{
        CReportCtrl& rc = GetReportCtrl();
        if(rc.GetStyle()&RVS_SHOWHGRIDEX)
              rc.ModifyStyle(RVS_SHOWHGRIDEX, 0);
        else
              rc.ModifyStyle(0, RVS_SHOWHGRID|RVS_SHOWHGRIDEX);
}

void CDisplayView::OnUpdateViewShowHGridEx(CCmdUI* pCmdUI)
{
        CReportCtrl& rc = GetReportCtrl();
        pCmdUI->SetCheck(rc.GetStyle()&RVS_SHOWHGRIDEX);
}

void CDisplayView::OnViewAlternateColors()
{
        CReportCtrl& rc = GetReportCtrl();
        if(rc.GetStyle()&RVS_SHOWCOLORALTERNATE)
              rc.ModifyStyle(RVS_SHOWCOLORALTERNATE, 0);
        else
              rc.ModifyStyle(0, RVS_SHOWCOLORALTERNATE);
```

```
}

void CDisplayView::OnUpdateViewAlternateColors(CCmdUI* pCmdUI)
{
      CReportCtrl& rc = GetReportCtrl();
      pCmdUI->SetCheck(rc.GetStyle()&RVS_SHOWCOLORALTERNATE);
}

void CDisplayView::OnNmReturn(NMHDR* pNMHDR, LRESULT* pResult)
{
      LPNMREPORTVIEW lpnmrv = (LPNMREPORTVIEW)pNMHDR;
      if (lpnmrv->nKeys == VK_RETURN) {
            ::PostMessage(GetParentFrame()->m_hWnd,WM_USER_PLAY,(WPARAM)
0, (LPARAM)0);
            *pResult = 0;
      }
}

DROPEFFECT CDisplayView::OnDragEnter (COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    CReportView::OnDragEnter (pDataObject, dwKeyState, point);

      if (pDataObject->IsDataAvailable (CF_HDROP))  {
            // Only Copy allowed on filenames
            return DROPEFFECT_COPY;
      }
      else
            return DROPEFFECT_NONE;
}

DROPEFFECT CDisplayView::OnDragOver (COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    CReportView::OnDragOver (pDataObject, dwKeyState, point);

    if (pDataObject->IsDataAvailable (CF_HDROP))  {
            // Only Copy allowed on filenames
            return DROPEFFECT_COPY;
      }
      else
            return DROPEFFECT_NONE;
}

BOOL CDisplayView::OnDrop (COleDataObject* pDataObject, DROPEFFECT
dropEffect,
    CPoint point)
{
    CReportView::OnDrop (pDataObject, dropEffect, point);

      // See if filenames are on the clipboard.
    HDROP hDrop = (HDROP) pDataObject->GetGlobalData (CF_HDROP);
    if (hDrop != NULL) {
            // Find out how many file names the HDROP contains.
        int nCount = ::DragQueryFile (hDrop, (UINT) -1, NULL, 0);
        // Enumerate the file names.
        if (nCount) {
            TCHAR szFile[MAX_PATH];

            CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>
(GetDocument());
```

```
                    for (int i=0; i<nCount; i++) {
                        ::DragQueryFile (hDrop, i, szFile,
                      sizeof (szFile) / sizeof (TCHAR));
                        pDoc->AddMediaToXML(currentFolderNode,
                                                    szFile,
                                                    "Local",
                                                    "-",
                                                    szFile,

      CUtils::get_currdatetime(),
                                                    "-",
                                                    "local",
                                                    "none");
                    }
                        pDoc->UpdateAllViews(NULL);
                }
                ::GlobalFree (hDrop);
                return TRUE;      // Drop succeeded.
        }
        return FALSE;           // Drop failed.
}


void CDisplayView::OnPlayMedia()
{
      ::PostMessage(AfxGetMainWnd()->m_hWnd,WM_USER_PLAY, 0, 0);
}
```

```
#if !defined(AFX_DISPLAYVIEW_H__A7AFBE0C_5E5C_4EC4_88C5_853C27172BF5
__INCLUDED_)
#define AFX_DISPLAYVIEW_H__A7AFBE0C_5E5C_4EC4_88C5_853C27172BF5
__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <libxml/tree.h>
#include <string>
#include <vector>
#include <afxole.h>
#include "ReportCtrl.h"

class CDisplayView : public CReportView
{
protected:
      DECLARE_DYNCREATE(CDisplayView)

// Attributes
public:
      CDisplayView();
// Operations
public:
      void AddMediaFromXML(xmlNodePtr node);

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDisplayView)
      public:
      virtual void OnInitialUpdate();
      protected:
      virtual void OnDraw(CDC* pDC);      // overridden to draw this
view
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);
    virtual DROPEFFECT OnDragEnter(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point);
    virtual DROPEFFECT OnDragOver(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point);
    virtual BOOL OnDrop(COleDataObject* pDataObject, DROPEFFECT
dropEffect, CPoint point);
      //}}AFX_VIRTUAL

// Implementation
protected:
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

      // Generated message map functions
protected:
      COleDropTarget m_oleDropTarget;
      CPoint rightClickPoint;
      bool SetupColumns();
      //{{AFX_MSG(CDisplayView)
      afx_msg void OnRclick(NMHDR* pNMHDR, LRESULT* pResult);
      afx_msg void OnDeleteMedia();
```

```
        afx_msg void OnPlayMedia();
        afx_msg void OnDblclk(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnClick(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnRvnItemCallback(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnRvnSelectionChanged(NMHDR* pNMHDR, LRESULT*
pResult);
        afx_msg void OnNmReturn(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnViewShowVGrid();
        afx_msg void OnUpdateViewShowVGrid(CCmdUI* pCmdUI);
        afx_msg void OnViewShowHGrid();
        afx_msg void OnUpdateViewShowHGrid(CCmdUI* pCmdUI);
        afx_msg void OnViewShowHGridEx();
        afx_msg void OnUpdateViewShowHGridEx(CCmdUI* pCmdUI);
        afx_msg void OnViewAlternateColors();
        afx_msg void OnUpdateViewAlternateColors(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        xmlNodePtr currentFolderNode;
        std::vector<xmlNodePtr> currentItemData;
        int m_iSelected;
};

class SShowSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

class SEpisodeSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

class SDescriptionSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

class SReceivedSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

class SPublishedSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

class SDaysToPurgeSort
{
public:
        bool operator()(xmlNodePtr& item1, xmlNodePtr& item2);
};

//////////////////////////////////////////////////////////////////////////
```

```
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_DISPLAYVIEW_H__A7AFBE0C_5E5C_4EC4_88C5_
853C27172BF5__INCLUDED_)
```

```
// DownloadThread.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "playerDoc.h"
#include "globals.h"
#include "DownloadThread.h"
#include "ServerConnection.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CDownloadThread

IMPLEMENT_DYNCREATE(CDownloadThread, CWinThread)

CDownloadThread::CDownloadThread()
{
      m_pDoc = NULL;
      m_stopThread = false;
      m_pMainHWnd = NULL;
      m_PCConn = NULL;
//    m_bAutoDelete = FALSE;
}

CDownloadThread::~CDownloadThread()
{
}

BOOL CDownloadThread::InitInstance()
{
      // TODO:  perform and per-thread initialization here
      return TRUE;
}

int CDownloadThread::ExitInstance()
{
      if(m_PCConn)
            delete m_PCConn;
      m_PCConn = NULL;

      return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CDownloadThread, CWinThread)
      //{{AFX_MSG_MAP(CDownloadThread)
            // NOTE - the ClassWizard will add and remove mapping macros
here.
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
//////
// CDownloadThread message handlers
```

```
int CDownloadThread::Run()
{
      // Check the local document for validity.
      // Exit Thread if error.
      if(m_pDoc == NULL) {
            TRACE0("DownloadThread Error:  PlayerDoc is NULL.");
            ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD, NULL,
DT_NULLPLAYERDOC);
            return ExitInstance();
      }

      // Check the HWnd object. Exit Thread if error.
      if(m_pMainHWnd == NULL) {
            TRACE0("DownloadThread Error:  MainHWnd is NULL.");
            ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD, NULL,
DT_NULLHWND);
            return ExitInstance();
      }

      // Establish a server connection.
      typedef vector<string>::iterator VI;
      m_PCConn = new CServerConnection(m_pMainHWnd);

      // Get the Ip Address and Port of the server.
      string ipAddress = m_pDoc->GetServerIP();
      int portNumber = m_pDoc->GetServerPortNum();
      // Make a connection to the server.
      m_PCConn->connectPC(ipAddress,portNumber);

      // Get the username, password and playerid.
      string username = m_pDoc->GetUsername();
      string password = m_pDoc->GetPassword();
      string playerid = m_pDoc->GetPlayerID();
      // Login into the server.
      m_PCConn->login(username, password, playerid);

      // Start the icon animation.
      ::PostMessage(m_pMainHWnd, WM_USER_NOTIFYICON_ANIMATION_START,
(LPARAM)0, (WPARAM)0);

      vector<string> v_eid;
      vector<string> v_sdesc;
      vector<string> v_edesc;
      vector<string> v_filename;
      vector<string> v_contenturls;
      vector<string> v_contentmhtmls;

      // Receive the Subscriptions from the server.
      m_PCConn->recvSubs(&v_eid, &v_sdesc, &v_edesc,
                         &v_filename, &v_contenturls,
                         &v_contentmhtmls);

      /*************************** NOTE ******************************/
      /* This is extrememly strange.  if the doc was just create
      */
      /* IE the program is running for the first time, you need the
      */
      /* first node, if its been run before you need the second.
      */
      /* The strange part is that the Folders view doesn't have this
```

```
      */
      /* problem.  I think it might be that when AddNode is called,
      */
      /* it doens't set next.  When its read from a file, it does.
      */
      xmlNodePtr node = m_pDoc->GetFolders()->children->children;
      if(node == NULL) node = m_pDoc->GetFolders()->children->next->
children;
      xmlNodePtr inboxNode = NULL;
      bool done = false;
      string strLabel;
      xmlNodePtr folder = node;
      TRACE1("ThreadFolder set to: %s\n", node->name);

      // Check for a valid inbox.
      // Exit thread if none exists.
      inboxNode = m_pDoc->GetInboxNodePtr();
      if(inboxNode == NULL) {
            TRACE0("DownloadThread Error:  Could not find inbox node.");
            ::PostMessage(m_pMainHWnd, WM_USER_NOTIFYICON_ANIMATION_END,
(LPARAM)0, (WPARAM)0);
            ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD, NULL,
DT_INBOXERROR);
            return ExitInstance();
      }

      time_t occurance;
      string curDateTime;
      occurance = time(NULL);
      struct tm *today;
      char tmpbuf[128];
      long int fsize;

      today = localtime(&occurance);
      strftime(tmpbuf, 128, "%a %m/%d/%y  %I:%M %p", today );
      curDateTime = tmpbuf;

      VI sdescIter = v_sdesc.begin();
      VI eidIter = v_eid.begin();
      VI edescIter = v_edesc.begin();
      VI curlIter = v_contenturls.begin();
      VI cmhtmlIter = v_contentmhtmls.begin();

      // Each iteration receives a file.
      // Stop the loop if m_stopThread is true.
      for(VI i = v_filename.begin(); ((!m_stopThread) && (i !=
v_filename.end())); i++) {
            string url = m_pDoc->GetDataDir();
            string mhtml_path = url;
            url = url +"\\" + (*i);
            if( ((string)*cmhtmlIter).length() > 0 ) {
                  mhtml_path = mhtml_path + "\\" + (*cmhtmlIter);
            }
            else mhtml_path = "none";
            // Get Num Bytes About to Come Down.
            if(!m_PCConn->getNumBytesOfNextFile(&fsize)) {
                  TRACE("DownloadThread Error:  couldn't get file size
of next file to download.");
                  ::PostMessage(m_pMainHWnd,
WM_USER_NOTIFYICON_ANIMATION_END, (LPARAM)0, (WPARAM)0);
                  ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD,
```

```
NULL, DT_NETWORKERROR);
                return ExitInstance();
            }
            // Verify its ok to take in that many bytes.
            if(!m_pDoc->HaveAvailDiskSpace((unsigned __int64) fsize)) {
                TRACE("DownloadThread Error:  Disk Space Too Low.");
                ::PostMessage(m_pMainHWnd,
WM_USER_NOTIFYICON_ANIMATION_END, (LPARAM)0, (WPARAM)0);
                ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD,
NULL, DT_DISKSPACEERROR);
                return ExitInstance();

            }

            // Get the media file.
            if(m_PCConn->recvFile(url,fsize)) {

                // Download the MHTML if available.
                if(((string)*cmhtmlIter).length() > 0) {
                    // Get Num Bytes About to Come Down.
                    if(!m_PCConn->getNumBytesOfNextFile(&fsize)) {
                        TRACE("DownloadThread Error:  couldn't get
file size of next file to download.");
                        ::PostMessage(m_pMainHWnd,
WM_USER_NOTIFYICON_ANIMATION_END, (LPARAM)0, (WPARAM)0);
                        ::PostMessage(m_pMainHWnd,
WM_USER_DONE_DOWNLOAD, NULL, DT_NETWORKERROR);
                        return ExitInstance();
                    }
                    // Make sure the user has enough disk space for
the file.
                    // Exit thread and give an error message to the
user if
                    // the disk space is too low
                    if(!m_pDoc->HaveAvailDiskSpace((unsigned
__int64) fsize)) {
                        MessageBox(m_pMainHWnd,"Disk space is low.
The Synchronizer has been halted.  It will not continue until disk space
is freed.","Alert!", MB_OK | MB_ICONEXCLAMATION);
                        TRACE("DownloadThread Error:  couldn't get
file size of next file to download.");
                        ::PostMessage(m_pMainHWnd,
WM_USER_NOTIFYICON_ANIMATION_END, (LPARAM)0, (WPARAM)0);
                        ::PostMessage(m_pMainHWnd,
WM_USER_DONE_DOWNLOAD, NULL, DT_DISKSPACEERROR);
                        return ExitInstance();

                    }
                    // Get the mhtml file.
                    m_PCConn->recvFile(mhtml_path,fsize);
                }

                // Add the xml node for the recently downloaded file.
                if(!m_stopThread) {
                    m_pDoc->AddMediaToXML(inboxNode, url,
*sdescIter, *eidIter,
                                                        *edescIter,
curDateTime, "-",
                                                        *curlIter,
mhtml_path);
                    m_pDoc->WriteXMLFoldersDoc();
```

```
                        ::PostMessage(m_pMainHWnd,
WM_USER_UPDATEALLVIEWS, (LPARAM)0, (WPARAM)0);
                    }
            }

            sdescIter++; eidIter++; edescIter++; curlIter++;
cmhtmlIter++;
      }

      // Upload the Stats file unless m_stopThread is true.
      if((!m_stopThread) && (m_PCConn->ulStats(m_pDoc->
GetStatsFileLocation()))) {
            /* clear the file. */
            FILE *sfd;
            sfd = fopen(m_pDoc->GetStatsFileLocation().c_str(), "wb+");
            fclose(sfd);
            // MFC stats.clear stats
            (m_pDoc->GetStats())->clearStats();
      }

      ::PostMessage(m_pMainHWnd, WM_USER_NOTIFYICON_ANIMATION_END,
(LPARAM)0, (WPARAM)0);

      if(m_stopThread) {
            ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD, NULL,
DT_DOWNLOADABORTED);
            return ExitInstance();
      }


      ::PostMessage(m_pMainHWnd, WM_USER_DONE_DOWNLOAD, NULL,
DT_DOWNLOADSUCCESSFUL);
      return ExitInstance();
}

// Purpose: Set the Player Document and   m_pMainWnd
//                to the passed in variables.
// Returns: DT_NOERROR if params are valid.
//                DT_PDOCNULL if pDoc is invalid.
//                DT_HWNDNULL if cWnd is invalid.
//                DT_PDOCNULL + DT_HWNDNULL if both pDoc and cWnd are
invalid.
int CDownloadThread::SetParameters(HWND hWnd, CPlayerDoc *pDoc)
{
      // Set vars.
      m_pDoc = pDoc;
      m_pMainHWnd = hWnd;

      // Initialize error variable.
      // Return false if parameters are invalid.
      int error = DT_NOERROR;

      if(!m_pDoc) error += DT_PDOCNULL;
      if(!m_pMainHWnd) error += DT_HWNDNULL;

      // Parameters were valid so return true.
      return error;
}

// Purpose: Set m_stopThread to stop the thread.
void CDownloadThread::KillThread() {
```

```
        m_stopThread = true;
        m_PCConn->stop();
    }
```

```
#if !defined(AFX_DOWNLOADTHREAD_H__6CCB86C8_42A2_494F_B096_278367C04AD0
__INCLUDED_)
#define AFX_DOWNLOADTHREAD_H__6CCB86C8_42A2_494F_B096_278367C04AD0
__INCLUDED_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DownloadThread.h : header file
//

// SetParameters Error Codes.
#define DT_NOERROR                      0
#define DT_PDOCNULL                     1
#define DT_HWNDNULL                     2
#define DT_PDOC_AND_HWND_NULL 3

// Thread Return Codes.
#define DT_DOWNLOADSUCCESSFUL  1
#define DT_DOWNLOADABORTED              0
#define DT_NULLPLAYERDOC               -1
#define DT_NULLHWND                    -2
#define DT_INBOXERROR                  -3
#define DT_DISKSPACEERROR              -4
#define DT_NETWORKERROR                -5

/////////////////////////////////////////////////////////////////////////////
/////
// CDownloadThread thread
class CPlayerDoc;

class CDownloadThread : public CWinThread
{
      DECLARE_DYNCREATE(CDownloadThread)
protected:
      CDownloadThread();

// Attributes
public:
      int SetParameters(HWND hWnd, CPlayerDoc* pDoc);
      void KillThread();

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDownloadThread)
      public:
      virtual BOOL InitInstance();
      virtual int ExitInstance();
      virtual int Run();
      //}}AFX_VIRTUAL

// Implementation
protected:
      virtual ~CDownloadThread();

      // Generated message map functions
      //{{AFX_MSG(CDownloadThread)
```

```
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_MSG

      DECLARE_MESSAGE_MAP()

private:
      CServerConnection* m_PCConn;
      HWND m_pMainHWnd;
      CPlayerDoc *m_pDoc;
      bool m_stopThread;
};

/////////////////////////////////////////////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_DOWNLOADTHREAD_H__6CCB86C8_42A2_494F_B096_
278367C04AD0__INCLUDED_)
```

```
// ErrorDlg.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "ErrorDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CErrorDlg dialog


CErrorDlg::CErrorDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CErrorDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CErrorDlg)
      m_msg = _T("");
      //}}AFX_DATA_INIT
}


void CErrorDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CErrorDlg)
      DDX_Control(pDX, IDC_EDIT1, m_msgCtrl);
      DDX_Text(pDX, IDC_EDIT1, m_msg);
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CErrorDlg, CDialog)
      //{{AFX_MSG_MAP(CErrorDlg)
            // NOTE: the ClassWizard will add message map macros here
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CErrorDlg message handlers
```

```
#if !defined(AFX_ERRORDLG_H__B30202B9_B782_48D6_AECA_D860727B6664
__INCLUDED_)
#define AFX_ERRORDLG_H__B30202B9_B782_48D6_AECA_D860727B6664__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ErrorDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
/////
// CErrorDlg dialog

class CErrorDlg : public CDialog
{
// Construction
public:
      CErrorDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CErrorDlg)
      enum { IDD = IDD_ERROR_DLG };
      CEdit m_msgCtrl;
      CString     m_msg;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CErrorDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CErrorDlg)
            // NOTE: the ClassWizard will add member functions here
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_ERRORDLG_H__B30202B9_B782_48D6_AECA_D860727B6664
__INCLUDED_)
```

```
////////////////////////////////////////////////////////////////////////
////
//    File:       CFlatHeaderCtrl.cpp
//    Version:    1.0.6
//
//    Author:           Maarten Hoeben
//    E-mail:           hoeben@nwn.com
//
//    Implementation of the CFlatHeaderCtrl and associated classes.
//
//    This code may be used in compiled form in any way you desire. This
//    file may be redistributed unmodified by any means PROVIDING it is
//    not sold for profit without the authors written consent, and
//    providing that this notice and the authors name and all copyright
//    notices remains intact.
//
//    An email letting me know how you are using it would be nice as
well.
//
//    This file is provided "as is" with no expressed or implied
warranty.
//    The author accepts no liability for any damage/loss of business
that
//    this product may cause.
//
//    Version history
//
//    1.0.0 - Initial release
//    1.0.1 - Fixed FHDragWnd destroy warning (thanks Philippe Terrier)
//                - Fixed double sent HDN_ITEMCLICK
//                - Added a property that adjusts for ListCtrls that use
a static
//                  border for flat look.
//    1.0.2 - Fixed another destroy warning
//                - Fixed InsertItem array exception handling
//                - Fixed incorrect header width painting
//                - Changed DrawItem argument passing
//                - Changed HDITEMEX struct item names
//                - Added handler for HDM_SETIMAGELIST (precalculate
image dimensions)
//                - Changed DrawImage to clip images
//                - Changed InsertItem ASSERT check to position
limitation
//                - Added new-style "HotDivider" arrows
//                - Fixed some GDI objects
//                - Added 'don't drop cursor' support to indicate
drag&drop
//                  outside control
//                - Added drag&drop target window support
//                - Changed CFHDragWnd to support externally created
items
//                - Removed topmost-style from CFHDropWnd
//                - Fixed OnSetHotDivider order bug
//                - Added extended styles
//                - Added item width estimation function
//    1.0.3 - Added WM_CANCELMODE handler
//    1.0.4 - Changed copyright message
//                - Added tooltip style
//    1.0.5 - Added persistent style
//                - Added definitions for drop result
//                - Added height manipulation functions
//
```

```
//    1.0.6 - Fixed bitmap drawing resource leak
//                - Added proper font handling.
//
////////////////////////////////////////////////////////////////////////////
////


// FlatHeaderCtrl.cpp : implementation file
//

#include "stdafx.h"
#include "FlatHeaderCtrl.h"

#include <afxpriv.h>

#include "MemDC.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////
/////
// CFHDragWnd

CFHDragWnd::CFHDragWnd()
{
      // Register the window class if it has not already been
registered.
      WNDCLASS wndclass;
      HINSTANCE hInst = AfxGetInstanceHandle();

      if(!(::GetClassInfo(hInst, FHDRAGWND_CLASSNAME, &wndclass)))
      {
            // otherwise we need to register a new class
            wndclass.style = CS_SAVEBITS ;
            wndclass.lpfnWndProc = ::DefWindowProc;
            wndclass.cbClsExtra = wndclass.cbWndExtra = 0;
            wndclass.hInstance = hInst;
            wndclass.hIcon = NULL;
            wndclass.hCursor = LoadCursor( hInst, IDC_ARROW);
            wndclass.hbrBackground = (HBRUSH)(COLOR_3DFACE + 1);
            wndclass.lpszMenuName = NULL;
            wndclass.lpszClassName = FHDRAGWND_CLASSNAME;
            if (!AfxRegisterClass(&wndclass))
                  AfxThrowResourceException();
      }

      m_pFlatHeaderCtrl = NULL;
      m_iItem = -1;
      m_lphdiItem = NULL;
}

CFHDragWnd::~CFHDragWnd()
{
}


BEGIN_MESSAGE_MAP(CFHDragWnd, CWnd)
```

```
    //{{AFX_MSG_MAP(CFHDragWnd)
    ON_WM_PAINT()
    ON_WM_ERASEBKGND()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CFHDragWnd message handlers

BOOL CFHDragWnd::Create(CRect rect, CFlatHeaderCtrl* pFlatHeaderCtrl,
INT iItem, LPHDITEM lphdiItem)
{
    ASSERT_VALID(pFlatHeaderCtrl);
    ASSERT(pFlatHeaderCtrl->IsKindOf(RUNTIME_CLASS(CFlatHeaderCtrl)));

    m_pFlatHeaderCtrl = pFlatHeaderCtrl;
    m_iItem = iItem;
    m_lphdiItem = lphdiItem;

    DWORD dwStyle = WS_POPUP|WS_DISABLED;
    DWORD dwExStyle = WS_EX_TOOLWINDOW|WS_EX_TOPMOST;

    return CreateEx(dwExStyle, FHDRAGWND_CLASSNAME, NULL, dwStyle,
        rect.left, rect.top, rect.Width(), rect.Height(),
        NULL, NULL, NULL );
}

void CFHDragWnd::OnPaint()
{
    CPaintDC dc(this);

    if(m_pFlatHeaderCtrl->m_bDoubleBuffer)
    {
        CMemDC MemDC(&dc);
        OnDraw(&MemDC);
    }
    else
        OnDraw(&dc);
}

BOOL CFHDragWnd::OnEraseBkgnd(CDC* pDC)
{
    return TRUE;
}

void CFHDragWnd::OnDraw(CDC* pDC)
{
    CRect rect;
    GetClientRect(rect);

    pDC->FillSolidRect(rect, m_pFlatHeaderCtrl->m_cr3DFace);
    pDC->Draw3dRect(rect, m_pFlatHeaderCtrl->m_cr3DHighLight,
m_pFlatHeaderCtrl->m_cr3DShadow);

    CPen* pPen = pDC->GetCurrentPen();
    CFont* pFont = pDC->SelectObject(m_pFlatHeaderCtrl->GetFont());

    pDC->SetBkColor(m_pFlatHeaderCtrl->m_cr3DFace);
```

```
        pDC->SetTextColor(m_pFlatHeaderCtrl->m_crText);

        rect.DeflateRect(m_pFlatHeaderCtrl->m_iSpacing, 0);
        m_pFlatHeaderCtrl->DrawItem(
              pDC,
              rect,
              m_lphdiItem,
              m_pFlatHeaderCtrl->m_iSortColumn == m_iItem,
              m_pFlatHeaderCtrl->m_bSortAscending
        );

        pDC->SelectObject(pFont);
        pDC->SelectObject(pPen);
  }

  void CFHDragWnd::PostNcDestroy()
  {
        CWnd::PostNcDestroy();
        delete this;
  }

  ////////////////////////////////////////////////////////////////////////////
  /////
  // CFHDropWnd

  CFHDropWnd::CFHDropWnd(COLORREF crColor)
  {
        m_brush.CreateSolidBrush(crColor);

        // Register the window class if it has not already been
  registered.
        WNDCLASS wndclass;
        HINSTANCE hInst = AfxGetInstanceHandle();

        if(!(::GetClassInfo(hInst, FHDROPWND_CLASSNAME, &wndclass)))
        {
              // otherwise we need to register a new class
              wndclass.style = CS_SAVEBITS ;
              wndclass.lpfnWndProc = ::DefWindowProc;
              wndclass.cbClsExtra = wndclass.cbWndExtra = 0;
              wndclass.hInstance = hInst;
              wndclass.hIcon = NULL;
              wndclass.hCursor = LoadCursor( hInst, IDC_ARROW );
              wndclass.hbrBackground = (HBRUSH)m_brush;
              wndclass.lpszMenuName = NULL;
              wndclass.lpszClassName = FHDROPWND_CLASSNAME;
              if (!AfxRegisterClass(&wndclass))
                    AfxThrowResourceException();
        }
  }

  CFHDropWnd::~CFHDropWnd()
  {
  }


  BEGIN_MESSAGE_MAP(CFHDropWnd, CWnd)
        //{{AFX_MSG_MAP(CFHDropWnd)
        ON_WM_ERASEBKGND()
        //}}AFX_MSG_MAP
  END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////
/////
// CFHDropWnd message handlers

BOOL CFHDropWnd::Create(INT iHeight)
{
      m_iHeight = iHeight + 20;

      DWORD dwStyle = WS_POPUP|WS_DISABLED;
      DWORD dwExStyle = WS_EX_TOOLWINDOW ;

      BOOL bResult = CreateEx(dwExStyle, FHDROPWND_CLASSNAME, NULL,
dwStyle,
            0, 0, 12, m_iHeight,
            NULL, NULL, NULL );

      CRgn rgn1, rgn2;
      POINT ptArrow[7];

      ptArrow[0].x = 8; ptArrow[0].y = 0;
      ptArrow[1].x = 8; ptArrow[1].y = 4;
      ptArrow[2].x = 11;      ptArrow[2].y = 4;
      ptArrow[3].x = 6; ptArrow[3].y = 9;
      ptArrow[4].x = 1; ptArrow[4].y = 4;
      ptArrow[5].x = 4; ptArrow[5].y = 4;
      ptArrow[6].x = 4; ptArrow[6].y = 0;
      rgn1.CreatePolygonRgn(ptArrow, 7, ALTERNATE);


      ptArrow[0].x = 4; ptArrow[0].y = m_iHeight;
      ptArrow[1].x = 4; ptArrow[1].y = m_iHeight-4;
      ptArrow[2].x = 0; ptArrow[2].y = m_iHeight-4;
      ptArrow[3].x = 6; ptArrow[3].y = m_iHeight-10;
      ptArrow[4].x = 12;      ptArrow[4].y = m_iHeight-4;
      ptArrow[5].x = 8; ptArrow[5].y = m_iHeight-4;
      ptArrow[6].x = 8; ptArrow[6].y = m_iHeight;
      rgn2.CreatePolygonRgn(ptArrow, 7, ALTERNATE);

      m_rgn.CreateRectRgn(0, 0, 12, iHeight);
      m_rgn.CombineRgn(&rgn1, &rgn2, RGN_OR);
      SetWindowRgn(m_rgn, FALSE);

      rgn1.DeleteObject();
      rgn2.DeleteObject();

      return bResult;
}


void CFHDropWnd::PostNcDestroy()
{
      m_rgn.DeleteObject();

      CWnd::PostNcDestroy();
      delete this;
}

BOOL CFHDropWnd::OnEraseBkgnd(CDC* pDC)
{
```

```
        pDC->FillRect(CRect(0, 0, 12, m_iHeight), &m_brush);
        return TRUE;
}

void CFHDropWnd::SetWindowPos(INT x, INT y)
{
        CWnd::SetWindowPos(
                &wndTop,
                x-6, y-(m_iHeight/2),
                0, 0, SWP_NOSIZE|SWP_SHOWWINDOW|SWP_NOACTIVATE
        );
}

////////////////////////////////////////////////////////////////////////////
/////
// CFlatHeaderCtrl

IMPLEMENT_DYNCREATE(CFlatHeaderCtrl, CHeaderCtrl)

CFlatHeaderCtrl::CFlatHeaderCtrl()
{
        m_bDoubleBuffer = TRUE;
        m_iSpacing = 6;
        m_sizeArrow.cx = 8;
        m_sizeArrow.cy = 8;
        m_sizeImage.cx = 0;
        m_sizeImage.cy = 0;
        m_bStaticBorder = FALSE;
        m_nDontDropCursor = 0;
        m_hDropTarget = NULL;
        m_rcDropTarget.SetRectEmpty();
        m_iDropResult = FHDR_ONHEADER;

        m_iHotIndex = -1;
        m_bHotItemResizable = FALSE;

        m_bResizing = FALSE;

        m_iHotDivider = -1;
        m_crHotDivider = 0x000000FF;
        m_pDropWnd = NULL;

        m_bDragging = FALSE;
        m_pDragWnd = NULL;

        m_nClickFlags = 0;

        m_bSortAscending = FALSE;
        m_iSortColumn = -1;
        m_arrayHdrItemEx.SetSize(0, 8);

        m_iHeight = -1;

        m_cr3DHighLight = ::GetSysColor(COLOR_3DHIGHLIGHT);
        m_cr3DShadow = ::GetSysColor(COLOR_3DSHADOW);
        m_cr3DFace = ::GetSysColor(COLOR_3DFACE);
        m_crText = ::GetSysColor(COLOR_BTNTEXT);

        m_font.CreateStockObject( DEFAULT_GUI_FONT );
}
```

```
CFlatHeaderCtrl::~CFlatHeaderCtrl()
{
    if(m_font.m_hObject)
            m_font.DeleteObject();

      if(m_pDropWnd != NULL)
      {
            m_pDropWnd->DestroyWindow();
            m_pDropWnd = NULL;
      }

      if(m_pDragWnd != NULL)
      {
            m_pDragWnd->DestroyWindow();
            m_pDragWnd = NULL;
      }
}

BEGIN_MESSAGE_MAP(CFlatHeaderCtrl, CHeaderCtrl)
      //{{AFX_MSG_MAP(CFlatHeaderCtrl)
      ON_MESSAGE(HDM_INSERTITEMA, OnInsertItem)
      ON_MESSAGE(HDM_INSERTITEMW, OnInsertItem)
      ON_MESSAGE(HDM_DELETEITEM, OnDeleteItem)
      ON_MESSAGE(HDM_SETIMAGELIST, OnSetImageList)
      ON_MESSAGE(HDM_SETHOTDIVIDER, OnSetHotDivider)
      ON_MESSAGE(HDM_LAYOUT, OnLayout)
      ON_NOTIFY_EX(TTN_NEEDTEXTA, 0, OnToolTipNotify)
      ON_NOTIFY_EX(TTN_NEEDTEXTW, 0, OnToolTipNotify)
      ON_WM_NCHITTEST()
      ON_WM_SETCURSOR()
      ON_WM_LBUTTONDOWN()
      ON_WM_LBUTTONDBLCLK()
      ON_WM_PAINT()
      ON_WM_SYSCOLORCHANGE()
      ON_WM_ERASEBKGND()
      ON_WM_LBUTTONUP()
      ON_WM_MOUSEMOVE()
      ON_WM_CANCELMODE()
      //}}AFX_MSG_MAP
    ON_MESSAGE(WM_SETFONT, OnSetFont)
    ON_MESSAGE(WM_GETFONT, OnGetFont)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
/////
// CFlatHeaderCtrl attributes

BOOL CFlatHeaderCtrl::ModifyProperty(WPARAM wParam, LPARAM lParam)
{
      switch(wParam)
      {
      case FH_PROPERTY_SPACING:
            m_iSpacing = (INT)lParam;
            break;

      case FH_PROPERTY_ARROW:
            m_sizeArrow.cx = LOWORD(lParam);
            m_sizeArrow.cy = HIWORD(lParam);
            break;

      case FH_PROPERTY_STATICBORDER:
```

```
            m_bStaticBorder = (BOOL)lParam;
            break;

      case FH_PROPERTY_DONTDROPCURSOR:
            m_nDontDropCursor = (UINT)lParam;
            break;

      case FH_PROPERTY_DROPTARGET:
            m_hDropTarget = (HWND)lParam;
            break;

      case FH_PROPERTY_ENABLETOOLTIPS:
            EnableToolTips((BOOL)lParam);
            break;

      default:
            return FALSE;
      }

      Invalidate();
      return TRUE;
}

BOOL CFlatHeaderCtrl::GetItemEx(INT iPos, HDITEMEX* phditemex) const
{
      if(iPos>=m_arrayHdrItemEx.GetSize())
            return FALSE;

      phditemex->nStyle = m_arrayHdrItemEx[iPos].nStyle;
      phditemex->iMinWidth = m_arrayHdrItemEx[iPos].iMinWidth;
      phditemex->iMaxWidth = m_arrayHdrItemEx[iPos].iMaxWidth;
      phditemex->strToolTip = m_arrayHdrItemEx[iPos].strToolTip;
      return TRUE;
}

BOOL CFlatHeaderCtrl::SetItemEx(INT iPos, HDITEMEX* phditemex)
{
      if(iPos>=m_arrayHdrItemEx.GetSize())
            return FALSE;

      BOOL bUpdate = FALSE;

      HDITEM hditem;
      hditem.mask = HDI_WIDTH;
      if(!GetItem(iPos, &hditem))
            return FALSE;

      HDITEMEX hditemex = *phditemex;

      if(hditemex.nStyle&HDF_EX_AUTOWIDTH)
      {
            TCHAR szText[FLATHEADER_TEXT_MAX];

            HDITEM hdi;
            hdi.mask =
HDI_WIDTH|HDI_FORMAT|HDI_TEXT|HDI_IMAGE|HDI_BITMAP;
            hdi.pszText = szText;
            hdi.cchTextMax = sizeof(szText);
            VERIFY(GetItem(iPos, &hdi));

            hditem.cxy = GetItemWidth(&hdi,
```

```
hditemex.nStyle&HDF_EX_INCLUDESORT ? TRUE:FALSE);
            bUpdate = TRUE;
      }

      if((!(hditemex.nStyle&HDF_EX_FIXEDWIDTH)) && (hditemex.iMinWidth
<=hditemex.iMaxWidth))
      {
            if(hditem.cxy < hditemex.iMinWidth)
            {
                  hditem.cxy = hditemex.iMinWidth;
                  bUpdate = TRUE;
            }

            if(hditem.cxy > hditemex.iMaxWidth)
            {
                  hditem.cxy = hditemex.iMaxWidth;
                  bUpdate = TRUE;
            }
      }

      if(bUpdate)
            SetItem(iPos, &hditem);

      m_arrayHdrItemEx.SetAt(iPos, hditemex);
      return TRUE;
}

INT CFlatHeaderCtrl::GetItemWidth(LPHDITEM lphdi, BOOL bIncludeSort)
{
      INT iWidth = 0;

      CBitmap* pBitmap = NULL;
      BITMAP biBitmap;
      if(lphdi->fmt&HDF_BITMAP)
      {
            ASSERT(lphdi->mask&HDI_BITMAP);
            ASSERT(lphdi->hbm);

            pBitmap = CBitmap::FromHandle(lphdi->hbm);
            if(pBitmap)
                  VERIFY(pBitmap->GetObject(sizeof(BITMAP), &biBitmap));
      }

      iWidth += m_iSpacing;
      iWidth += lphdi->fmt&HDF_IMAGE ? m_sizeImage.cx+m_iSpacing:0;
      iWidth += lphdi->fmt&HDF_BITMAP ? biBitmap.bmWidth+m_iSpacing:0;
      iWidth += bIncludeSort ? m_sizeArrow.cx+m_iSpacing:0;

      if(lphdi->mask&HDI_TEXT && lphdi->fmt&HDF_STRING)
      {
            CClientDC dc(this);
            CFont* pFont = dc.SelectObject(GetFont());

            iWidth += dc.GetTextExtent(lphdi->pszText).cx + m_iSpacing;

            dc.SelectObject(pFont);
      }

      return iWidth;
}
```

```
void CFlatHeaderCtrl::SetSortColumn(INT iPos, BOOL bSortAscending)
{
      ASSERT(iPos < GetItemCount());

      m_bSortAscending = bSortAscending;
      m_iSortColumn = iPos;
      Invalidate();
}

INT CFlatHeaderCtrl::GetSortColumn(BOOL* pbSortAscending)
{
      if(pbSortAscending)
            *pbSortAscending = m_bSortAscending;

      return m_iSortColumn;
}

INT CFlatHeaderCtrl::GetDropResult()
{
      return m_iDropResult;
}

INT CFlatHeaderCtrl::GetHeight()
{
      if(m_iHeight <= 0)
      {
            CRect rect;
            GetWindowRect(rect);

            return rect.Height();
      }
      else
            return m_iHeight;
}

void CFlatHeaderCtrl::SetHeight(INT iHeight)
{
      m_iHeight = iHeight;
}

//////////////////////////////////////////////////////////////////////////
/////
// CFlatHeaderCtrl implementation

void CFlatHeaderCtrl::DrawCtrl(CDC* pDC)
{
      CRect rectClip;
      if (pDC->GetClipBox(&rectClip) == ERROR)
            return;

      CRect rectClient, rectItem;
      GetClientRect(&rectClient);

    pDC->FillSolidRect(rectClip, m_cr3DFace);

      INT iItems = GetItemCount();
      ASSERT(iItems >= 0);

      CPen penHighLight(PS_SOLID, 1, m_cr3DHighLight);
      CPen penShadow(PS_SOLID, 1, m_cr3DShadow);
      CPen* pPen = pDC->GetCurrentPen();
```

```
        CFont* pFont = pDC->SelectObject(GetFont());

        pDC->SetBkColor(m_cr3DFace);
        pDC->SetTextColor(m_crText);

        INT iWidth = 0;

        for(INT i=0;i<iItems;i++)
        {
                INT iItem = OrderToIndex(i);

                TCHAR szText[FLATHEADER_TEXT_MAX];

                HDITEM hditem;
                hditem.mask =
HDI_WIDTH|HDI_FORMAT|HDI_TEXT|HDI_IMAGE|HDI_BITMAP;
                hditem.pszText = szText;
                hditem.cchTextMax = sizeof(szText);
                VERIFY(GetItem(iItem, &hditem));

                VERIFY(GetItemRect(iItem, rectItem));

                if (rectItem.right >= rectClip.left || rectItem.left <=
rectClip.right)
                {
                        if(hditem.fmt&HDF_OWNERDRAW)
                        {
                                DRAWITEMSTRUCT disItem;
                                disItem.CtlType = ODT_BUTTON;
                                disItem.CtlID = GetDlgCtrlID();
                                disItem.itemID = iItem;
                                disItem.itemAction = ODA_DRAWENTIRE;
                                disItem.itemState = 0;
                                disItem.hwndItem = m_hWnd;
                                disItem.hDC = pDC->m_hDC;
                                disItem.rcItem = rectItem;
                                disItem.itemData = 0;

                                DrawItem(&disItem);
                        }
                        else
                        {
                                rectItem.DeflateRect(m_iSpacing, 0);
                                DrawItem(pDC, rectItem, &hditem, iItem ==
m_iSortColumn, m_bSortAscending);
                                rectItem.InflateRect(m_iSpacing, 0);

                                if(m_nClickFlags&MK_LBUTTON && m_iHotIndex ==
iItem && m_hdhtiHotItem.flags&HHT_ONHEADER)
                                        pDC->InvertRect(rectItem);
                        }

                        if(i < iItems-1)
                        {
                                pDC->SelectObject(&penShadow);
                                pDC->MoveTo(rectItem.right-1, rectItem.top+2);
                                pDC->LineTo(rectItem.right-1, rectItem.bottom-
2);

                                pDC->SelectObject(&penHighLight);
```

```
                        pDC->MoveTo(rectItem.right, rectItem.top+2);
                        pDC->LineTo(rectItem.right, rectItem.bottom-2);
                  }
            }

            iWidth += hditem.cxy;
      }

      if(iWidth > 0)
      {
            rectClient.right = rectClient.left + iWidth;
            pDC->Draw3dRect(rectClient, m_cr3DHighLight, m_cr3DShadow);
      }

      pDC->SelectObject(pFont);
      pDC->SelectObject(pPen);

      penHighLight.DeleteObject();
      penShadow.DeleteObject();
}

void CFlatHeaderCtrl::DrawItem(LPDRAWITEMSTRUCT)
{
      ASSERT(FALSE);  // must override for self draw header controls
}

void CFlatHeaderCtrl::DrawItem(CDC* pDC, CRect rect, LPHDITEM lphdi,
BOOL bSort, BOOL bSortAscending)
{
      ASSERT(lphdi->mask&HDI_FORMAT);

      INT iWidth = 0;

      CBitmap* pBitmap = NULL;
      BITMAP BitmapInfo;
      if(lphdi->fmt&HDF_BITMAP)
      {
            ASSERT(lphdi->mask&HDI_BITMAP);
            ASSERT(lphdi->hbm);

            pBitmap = CBitmap::FromHandle(lphdi->hbm);
            if(pBitmap)
                  VERIFY(pBitmap->GetObject(sizeof(BITMAP),
&BitmapInfo));
      }

      switch(lphdi->fmt&HDF_JUSTIFYMASK)
      {
      case HDF_LEFT:
            rect.left += (iWidth = DrawImage(pDC, rect, lphdi, FALSE)) ?
iWidth+m_iSpacing : 0;
            if(lphdi->fmt&HDF_IMAGE && !iWidth)
                  break;
            rect.right -= bSort ? m_iSpacing+m_sizeArrow.cx : 0;
            rect.left += (iWidth = DrawText(pDC, rect, lphdi)) ?
iWidth+m_iSpacing : 0;
            if(bSort)
            {
                  rect.right += m_iSpacing+m_sizeArrow.cx;
                  rect.left += DrawArrow(pDC, rect, bSortAscending,
FALSE)+m_iSpacing;
```

```
            }
            DrawBitmap(pDC, rect, lphdi, pBitmap, &BitmapInfo, TRUE);
            break;

      case HDF_CENTER:
            rect.left += (iWidth = DrawImage(pDC, rect, lphdi, FALSE)) ?
iWidth+m_iSpacing : 0;
            if(lphdi->fmt&HDF_IMAGE && !iWidth)
                  break;

            rect.left += bSort ? m_iSpacing+m_sizeArrow.cx : 0;
            rect.right -= (iWidth=DrawBitmap(pDC, rect, lphdi, pBitmap,
&BitmapInfo, TRUE)) ? iWidth+m_iSpacing:0;
            if(bSort)
            {
                  rect.left -= m_iSpacing+m_sizeArrow.cx;
                  rect.right -= DrawArrow(pDC, rect, bSortAscending,
TRUE)+2*m_iSpacing;
            }
            DrawText(pDC, rect, lphdi);
            break;

       case HDF_RIGHT:
            if(!(lphdi->fmt&HDF_BITMAP_ON_RIGHT))
                  rect.left += (iWidth=DrawBitmap(pDC, rect, lphdi,
pBitmap, &BitmapInfo, FALSE)) ? iWidth+m_iSpacing:0;

            rect.left += (iWidth = DrawImage(pDC, rect, lphdi, FALSE)) ?
iWidth+m_iSpacing : 0;
            if(lphdi->fmt&HDF_IMAGE && !iWidth)
                  break;

            rect.left += bSort && (lphdi->fmt&HDF_BITMAP_ON_RIGHT) ?
m_iSpacing+m_sizeArrow.cx : 0;
            if(lphdi->fmt&HDF_BITMAP_ON_RIGHT)
                  rect.right -= (iWidth=DrawBitmap(pDC, rect, lphdi,
pBitmap, &BitmapInfo, TRUE)) ? iWidth+m_iSpacing:0;
            if(bSort)
            {
                  rect.left -= (lphdi->fmt&HDF_BITMAP_ON_RIGHT) ?
m_iSpacing+m_sizeArrow.cx:0;
                  rect.right -= DrawArrow(pDC, rect, bSortAscending,
TRUE)+2*m_iSpacing;
            }
            DrawText(pDC, rect, lphdi);
            break;
      }
}

INT CFlatHeaderCtrl::DrawImage(CDC* pDC, CRect rect, LPHDITEM lphdi,
BOOL bRight)
{
      CImageList* pImageList = GetImageList();
      INT iWidth = 0;

      if(lphdi->mask&HDI_IMAGE && lphdi->fmt&HDF_IMAGE)
      {
            ASSERT(pImageList);
            ASSERT(lphdi->iImage>=0 && lphdi->iImage<pImageList->
GetImageCount());
```

```
            if(rect.Width()>0)
            {
                  POINT point;

                  point.y = rect.CenterPoint().y - (m_sizeImage.cy>>1);

                  if(bRight)
                        point.x = rect.right - m_sizeImage.cx;
                  else
                        point.x = rect.left;

                  SIZE size;
                  size.cx = rect.Width()<m_sizeImage.cx ? rect.Width
()):m_sizeImage.cx;
                  size.cy = m_sizeImage.cy;
                  pImageList->DrawIndirect(pDC, lphdi->iImage, point,
size, CPoint(0, 0));

                  iWidth = m_sizeImage.cx;
            }
      }

      return iWidth;
}

INT CFlatHeaderCtrl::DrawBitmap(CDC* pDC, CRect rect, LPHDITEM lphdi,
CBitmap* pBitmap, BITMAP* pBitmapInfo, BOOL bRight)
{
      INT iWidth = 0;

      if(pBitmap)
      {
            iWidth = pBitmapInfo->bmWidth;
            if(iWidth<=rect.Width() && rect.Width()>0)
            {
                  POINT point;

                  point.y = rect.CenterPoint().y - (pBitmapInfo->
bmHeight>>1);

                  if(bRight)
                        point.x = rect.right - iWidth;
                  else
                        point.x = rect.left;

                  CDC dc;
                  if(dc.CreateCompatibleDC(pDC) == TRUE)
                  {
                        CBitmap* pBitmapDC = (CBitmap*)dc.SelectObject
(pBitmap);
                        ASSERT(pBitmapDC);

                        iWidth = pDC->BitBlt(
                              point.x, point.y,
                              pBitmapInfo->bmWidth, pBitmapInfo->
bmHeight,
                              &dc, 0, 0, SRCCOPY
                        ) ? iWidth:0;

                        dc.SelectObject(pBitmapDC);
                  }
```

```
                else
                        iWidth = 0;
            }
            else
                iWidth = 0;
        }

        return iWidth;
}

INT CFlatHeaderCtrl::DrawText(CDC* pDC, CRect rect, LPHDITEM lphdi)
{
        CSize size;

        if(rect.Width()>0 && lphdi->mask&HDI_TEXT && lphdi->
fmt&HDF_STRING)
        {
                size = pDC->GetTextExtent(lphdi->pszText);

                switch(lphdi->fmt&HDF_JUSTIFYMASK)
                {
                case HDF_LEFT:
                case HDF_LEFT|HDF_RTLREADING:
                        pDC->DrawText(lphdi->pszText, -1, rect,
DT_LEFT|DT_END_ELLIPSIS|DT_SINGLELINE|DT_VCENTER);
                        break;
                case HDF_CENTER:
                case HDF_CENTER|HDF_RTLREADING:
                        pDC->DrawText(lphdi->pszText, -1, rect,
DT_CENTER|DT_END_ELLIPSIS|DT_SINGLELINE|DT_VCENTER);
                        break;
                case HDF_RIGHT:
                case HDF_RIGHT|HDF_RTLREADING:
                        pDC->DrawText(lphdi->pszText, -1, rect,
DT_RIGHT|DT_END_ELLIPSIS|DT_SINGLELINE|DT_VCENTER);
                        break;
                }
        }

        size.cx = rect.Width()>size.cx ? size.cx:rect.Width();
        return size.cx>0 ? size.cx:0;
}

INT CFlatHeaderCtrl::DrawArrow(CDC* pDC, CRect rect, BOOL
bSortAscending, BOOL bRight)
{
        INT iWidth = 0;

        if(rect.Width()>0 && m_sizeArrow.cx<=rect.Width())
        {
                iWidth = m_sizeArrow.cx;

                rect.top += (rect.Height() - m_sizeArrow.cy - 1)>>1;
                rect.bottom = rect.top + m_sizeArrow.cy - 1;

                rect.left = bRight ? rect.right-m_sizeArrow.cy:rect.left;

                // Set up pens to use for drawing the triangle
                CPen penLight(PS_SOLID, 1, m_cr3DHighLight);
                CPen penShadow(PS_SOLID, 1, m_cr3DShadow);
                CPen *pPen = pDC->SelectObject(&penLight);
```

```
            if(bSortAscending)
            {
                  // Draw triangle pointing upwards
                  pDC->MoveTo(rect.left + ((m_sizeArrow.cx-1)>>1) + 1,
      rect.top);
                  pDC->LineTo(rect.left +  (m_sizeArrow.cx-1),
      rect.top + m_sizeArrow.cy - 1);
                  pDC->LineTo(rect.left,
                  rect.top + m_sizeArrow.cy - 1);

                  pDC->SelectObject(&penShadow);
                  pDC->MoveTo(rect.left + ((m_sizeArrow.cx-1)>>1),
      rect.top);
                  pDC->LineTo(rect.left,
            rect.top + m_sizeArrow.cy - 1);
            }
            else
            {
                  // Draw triangle pointing downwards
                  pDC->MoveTo(rect.left + ((m_sizeArrow.cx-1)>>1)+1,
      rect.top + m_sizeArrow.cy - 1);
                  pDC->LineTo(rect.left +  (m_sizeArrow.cx-1),
      rect.top);

                  pDC->SelectObject(&penShadow);
                  pDC->MoveTo(rect.left + ((m_sizeArrow.cx-1)>>1),
      rect.top + m_sizeArrow.cy - 1);
                  pDC->LineTo(rect.left,
            rect.top);
                  pDC->LineTo(rect.left + m_sizeArrow.cx,
      rect.top);
            }

            // Restore the pen
            pDC->SelectObject(pPen);

            penLight.DeleteObject();
            penShadow.DeleteObject();
      }

      return iWidth;
}

INT CFlatHeaderCtrl::OnToolHitTest(CPoint point, TOOLINFO* pTI) const
{
      INT iResult = CHeaderCtrl::OnToolHitTest(point, pTI);
      if(iResult != -1)
            return iResult;

      HDHITTESTINFO hdhti;
      hdhti.pt = point;
      iResult = ::SendMessage(GetSafeHwnd(), HDM_HITTEST, 0, (LPARAM)
(&hdhti));
      if(iResult > -1)
      {
            GetItemRect(iResult, &pTI->rect);

            pTI->cbSize = sizeof(TOOLINFO);
            pTI->hwnd = GetSafeHwnd();
            pTI->uFlags = TTF_ALWAYSTIP;
```

```
            pTI->lpszText = LPSTR_TEXTCALLBACK;
            pTI->uId = FLATHEADER_TT_MAGIC;
      }

      return iResult;
}

//////////////////////////////////////////////////////////////////////////
/////
// CHeaderCtrl message handlers

LRESULT CFlatHeaderCtrl::OnInsertItem(WPARAM wParam, LPARAM lParam)
{
      HDITEMEX hditemex;
      hditemex.iMinWidth = 0;
      hditemex.iMaxWidth = -1;

      LRESULT lResult = -1;

      WORD wItems = m_arrayHdrItemEx.GetSize();
      wParam = wParam<=wItems ? wParam:wItems;

      try
      {
            m_arrayHdrItemEx.InsertAt(wParam, hditemex);

            lResult = Default();
            if(lResult < 0)    // Cleanup
                  m_arrayHdrItemEx.RemoveAt(wParam);
      }
      catch(CMemoryException* e)
      {
            e->Delete();
      }

      return lResult;
}

LRESULT CFlatHeaderCtrl::OnDeleteItem(WPARAM wParam, LPARAM lParam)
{
      ASSERT((INT)wParam < m_arrayHdrItemEx.GetSize());
      m_arrayHdrItemEx.RemoveAt(wParam);

      return Default();
}

LRESULT CFlatHeaderCtrl::OnSetImageList(WPARAM wParam, LPARAM lParam)
{
      CImageList* pImageList;
      pImageList = CImageList::FromHandle((HIMAGELIST)lParam);

      IMAGEINFO info;
      if(pImageList->GetImageInfo(0, &info))
      {
            m_sizeImage.cx = info.rcImage.right - info.rcImage.left;
            m_sizeImage.cy = info.rcImage.bottom - info.rcImage.top;
      }

      return Default();
}
```

```
LRESULT CFlatHeaderCtrl::OnSetHotDivider(WPARAM wParam, LPARAM lParam)
{
	if(wParam)
	{
		HDHITTESTINFO hdhti;
		hdhti.pt.x = LOWORD(lParam);
		hdhti.pt.y = HIWORD(lParam);
		ScreenToClient(&hdhti.pt);

		INT iHotIndex = SendMessage(HDM_HITTEST, 0, (LPARAM)
(&hdhti));
		if(iHotIndex >= 0)
		{
			HDITEM hditem;
			hditem.mask = HDI_ORDER;
			VERIFY(GetItem(iHotIndex, &hditem));
			m_iHotDivider = hditem.iOrder;

			CRect rectItem;
			VERIFY(GetItemRect(iHotIndex, rectItem));
			if(hdhti.pt.x > rectItem.CenterPoint().x)
				m_iHotDivider++;
		}
		else
			m_iHotDivider = -1;
	}
	else
		m_iHotDivider = (INT)lParam;

	RECT rect;
	GetClientRect(&rect);

	INT iItems = GetItemCount();
	if(m_iHotDivider >= 0 && m_iHotDivider<=iItems+1)
	{
		if(m_pDropWnd == NULL)
		{
			m_pDropWnd = new CFHDropWnd(m_crHotDivider);
			if(m_pDropWnd)
				m_pDropWnd->Create(rect.bottom - rect.top);
		}

		if(m_pDropWnd != NULL)
		{
			POINT pt;
			pt.y = (rect.bottom-rect.top)/2;

			if(m_iHotDivider<iItems)
			{
				GetItemRect(OrderToIndex(m_iHotDivider), &rect);
				pt.x = rect.left - 1;
			}
			else
			{
				GetItemRect(OrderToIndex(iItems-1), &rect);
				pt.x = rect.right;
			}

			ClientToScreen(&pt);
			m_pDropWnd->SetWindowPos(pt.x, pt.y);
		}
```

```
        }
        else
        {
                if(m_pDropWnd != NULL)
                {
                        m_pDropWnd->DestroyWindow();
                        m_pDropWnd = NULL;
                }
        }

        return(LRESULT)m_iHotDivider;
}

LRESULT CFlatHeaderCtrl::OnLayout(WPARAM wParam, LPARAM lParam)
{
        LPHDLAYOUT lphdlayout = (LPHDLAYOUT)lParam;

        if(m_bStaticBorder)
                lphdlayout->prc->right += GetSystemMetrics(SM_CXBORDER)*2;

        LRESULT lResult = CHeaderCtrl::DefWindowProc(HDM_LAYOUT, 0,
lParam);

        if(m_iHeight > 0)
                lphdlayout->pwpos->cy = m_iHeight;

        return lResult;
}

BOOL CFlatHeaderCtrl::OnToolTipNotify(UINT nId, NMHDR *pNMHDR, LRESULT
*pResult)
{
        TOOLTIPTEXT *pTTT = (TOOLTIPTEXT *)pNMHDR;

        if(
                pNMHDR->idFrom == (UINT)FLATHEADER_TT_MAGIC &&
                !m_arrayHdrItemEx[m_iHotIndex].strToolTip.IsEmpty()
        ) {
                USES_CONVERSION;

                wcscpy((WCHAR*)pTTT->lpszText, A2W((LPCTSTR)m_arrayHdrItemEx
[m_iHotIndex].strToolTip));
                pTTT->lpszText = pTTT->szText;
                return TRUE;
        }
        else
                return FALSE;
}

////////////////////////////////////////////////////////////////////////////
/////
// CFlatHeaderCtrl message handlers

void CFlatHeaderCtrl::OnSysColorChange()
{
        CHeaderCtrl::OnSysColorChange();

        m_cr3DHighLight = ::GetSysColor(COLOR_3DHIGHLIGHT);
        m_cr3DShadow = ::GetSysColor(COLOR_3DSHADOW);
        m_cr3DFace = ::GetSysColor(COLOR_3DFACE);
        m_crText = ::GetSysColor(COLOR_BTNTEXT);
```

```
}

LRESULT CFlatHeaderCtrl::OnGetFont(WPARAM wParam, LPARAM lParam)
{
    return (LRESULT)m_font.m_hObject;
}

LRESULT CFlatHeaderCtrl::OnSetFont(WPARAM wParam, LPARAM lParam)
{
      LRESULT lResult = Default();

      CFont *pFont = CFont::FromHandle((HFONT)wParam);
      if(pFont)
      {
            LOGFONT lf;
            pFont->GetLogFont(&lf);

            m_font.DeleteObject();
            m_font.CreateFontIndirect(&lf);
      }

      return lResult;
}

BOOL CFlatHeaderCtrl::OnEraseBkgnd(CDC* pDC)
{
      return TRUE;
}

void CFlatHeaderCtrl::OnPaint()
{
    CPaintDC dc(this);

    if (m_bDoubleBuffer)
    {
        CMemDC MemDC(&dc);
        DrawCtrl(&MemDC);
    }
    else
        DrawCtrl(&dc);
}

UINT CFlatHeaderCtrl::OnNcHitTest(CPoint point)
{
      m_hdhtiHotItem.pt = point;
      ScreenToClient(&m_hdhtiHotItem.pt);

      m_iHotIndex = SendMessage(HDM_HITTEST, 0, (LPARAM)
(&m_hdhtiHotItem));
      if(m_iHotIndex >= 0)
      {
            HDITEM hditem;
            hditem.mask = HDI_ORDER;
            VERIFY(GetItem(m_iHotIndex, &hditem));
            m_iHotOrder = hditem.iOrder;

            HDITEMEX hditemex;
            if(GetItemEx(m_iHotIndex, &hditemex))
                  m_bHotItemResizable =
hditemex.nStyle&HDF_EX_FIXEDWIDTH ? FALSE:TRUE;
      }
```

```
        return CHeaderCtrl::OnNcHitTest(point);
}

BOOL CFlatHeaderCtrl::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
{
        if(m_iHotIndex>=0 && m_hdhtiHotItem.flags&
(HHT_ONDIVIDER|HHT_ONDIVOPEN) && !m_bHotItemResizable)
        {
                SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
                return TRUE;
        }

        return CHeaderCtrl::OnSetCursor(pWnd, nHitTest, message);
}

void CFlatHeaderCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
        m_nClickFlags = nFlags;
        m_ptClickPoint = point;

        if(m_iHotIndex >= 0)
        {
                m_hdiHotItem.mask =
HDI_WIDTH|HDI_FORMAT|HDI_TEXT|HDI_IMAGE|HDI_BITMAP|HDI_ORDER|HDI_LPARAM;
                m_hdiHotItem.pszText = m_szHotItemText;
                m_hdiHotItem.cchTextMax = sizeof(m_szHotItemText);
                VERIFY(GetItem(m_iHotIndex, &m_hdiHotItem));
                VERIFY(GetItemEx(m_iHotIndex, &m_hdieHotItem));

                if(m_hdhtiHotItem.flags&HHT_ONHEADER)
                {
                        RECT rectItem;
                        VERIFY(GetItemRect(m_iHotIndex, &rectItem));
                        InvalidateRect(&rectItem);
                }

                if(m_hdhtiHotItem.flags&(HHT_ONDIVIDER|HHT_ONDIVOPEN))
                {
                        if(!m_bHotItemResizable)
                                return;

                        HDITEMEX hditemex;
                        VERIFY(GetItemEx(m_iHotIndex, &hditemex));

                        CRect rectItem;
                        GetItemRect(m_iHotIndex, rectItem);
                        ClientToScreen(rectItem);

                        if(hditemex.iMinWidth <= hditemex.iMaxWidth)
                        {
                                CRect rectClip;
                                GetClipCursor(rectClip);

                                POINT point;
                                GetCursorPos(&point);

                                INT iOffset = point.x - rectItem.right;
```

```
                        rectClip.left = rectItem.left +
hditemex.iMinWidth + iOffset;
                        rectClip.right = rectItem.left +
hditemex.iMaxWidth + iOffset;

                        ClipCursor(rectClip);
                    }

                    m_bResizing = TRUE;
                }
            }

            CHeaderCtrl::OnLButtonDown(nFlags, point);
}


void CFlatHeaderCtrl::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    if(m_iHotIndex>=0 && m_hdhtiHotItem.flags&
(HHT_ONDIVIDER|HHT_ONDIVOPEN) && !m_bHotItemResizable)
        return;

    CHeaderCtrl::OnLButtonDblClk(nFlags, point);
}

void CFlatHeaderCtrl::OnLButtonUp(UINT nFlags, CPoint point)
{
    m_nClickFlags = nFlags;
    m_ptClickPoint = point;

    if(m_iHotIndex >= 0)
    {
        CWnd* pWnd = GetParent();

        if(m_hdhtiHotItem.flags&(HHT_ONDIVIDER|HHT_ONDIVOPEN))
        {
            if(m_bResizing)
            {
                ClipCursor(NULL);
                m_bResizing = FALSE;
            }
        }

        if(m_hdhtiHotItem.flags&HHT_ONHEADER)
        {
            if(m_bDragging)
            {
                NMHEADER nmhdr;
                nmhdr.hdr.hwndFrom = m_hWnd;
                nmhdr.hdr.idFrom = GetDlgCtrlID();
                nmhdr.hdr.code = HDN_ENDDRAG;
                nmhdr.iItem = m_iHotIndex;
                nmhdr.iButton =   0;
                nmhdr.pitem = &m_hdiHotItem;
                if(pWnd->SendMessage(WM_NOTIFY, 0, (LPARAM)
&nmhdr)==FALSE && m_iHotDivider>=0)
                {
                    INT iCount = GetItemCount();
                    ASSERT(m_iHotOrder < iCount);
                    ASSERT(m_iHotDivider <= iCount);
```

```
                                LPINT piArray = new INT[iCount*2];
                                if(piArray)
                                {
                                      GetOrderArray((LPINT)piArray,
iCount);

                                      for(INT i=0,j=0;i<iCount;i++)
                                      {
                                            if(j == m_iHotOrder)
                                                  j++;

                                            if(
                                                  (m_iHotOrder
<m_iHotDivider && i == m_iHotDivider-1) ||
                                                  (m_iHotOrder>
=m_iHotDivider && i == m_iHotDivider)
                                            )
                                                  piArray[iCount+i] =
piArray[m_iHotOrder];
                                            else
                                                  piArray[iCount+i] =
piArray[j++];
                                      }

                                      SetOrderArray(iCount, (LPINT)
&piArray[iCount]);
                                      delete piArray;
                                }
                                else
                                      AfxThrowMemoryException();
                          }

                          if(m_pDragWnd != NULL)
                          {
                                m_pDragWnd->DestroyWindow();
                                m_pDragWnd = NULL;
                          }

                          if (GetCapture()->GetSafeHwnd() == GetSafeHwnd
())
                                ReleaseCapture();

                          m_bDragging = FALSE;
                          OnSetHotDivider(FALSE, -1);

                          Invalidate();
                    }
                    else
                    {
                          RECT rectItem;
                          VERIFY(GetItemRect(m_iHotIndex, &rectItem));
                          InvalidateRect(&rectItem);
                    }
              }
        }

        CHeaderCtrl::OnLButtonUp(nFlags, point);
  }

  void CFlatHeaderCtrl::OnMouseMove(UINT nFlags, CPoint point)
  {
```

```
	if(m_nClickFlags&MK_LBUTTON && m_iHotIndex>=0)
	{
		if(m_bResizing)
			CHeaderCtrl::OnMouseMove(nFlags, point);

		if(m_hdhtiHotItem.flags&HHT_ONHEADER)
		{
			if(m_bDragging)
			{
				if(m_pDragWnd != NULL)
				{
					CRect rect;
					m_pDragWnd->GetWindowRect(&rect);

					CPoint pt = point;
					ClientToScreen(&pt);

					pt.Offset(-(rect.Width()>>1), -
(rect.Height()>>1));

					m_pDragWnd->SetWindowPos(
						&wndTop,
						pt.x, pt.y,
						0, 0,
SWP_NOSIZE|SWP_SHOWWINDOW|SWP_NOACTIVATE
					);

					HDHITTESTINFO hdhti;
					hdhti.pt.x = point.x;
					hdhti.pt.y = point.y;

					INT iHotOrder = -1;
					INT iHotIndex = SendMessage(HDM_HITTEST,
0, (LPARAM)(&hdhti));
					if(iHotIndex >= 0)
					{
						HDITEM hditem;
						hditem.mask = HDI_ORDER;
						VERIFY(GetItem(iHotIndex, &hditem));
						iHotOrder = hditem.iOrder;

						CRect rectItem;
						VERIFY(GetItemRect(iHotIndex,
rectItem));
						if(hdhti.pt.x > rectItem.CenterPoint
().x)
							iHotOrder++;

						SetCursor(AfxGetApp()->
LoadStandardCursor(IDC_ARROW));
						m_iDropResult = FHDR_ONHEADER;
					}
					else
					{
						pt = point;
						ClientToScreen(&pt);

						if
(m_hdieHotItem.nStyle&HDF_EX_PERSISTENT)
						{
							SetCursor(AfxGetApp()->
```

```
LoadStandardCursor(IDC_NO));                    m_iDropResult =
FHDR_PERSISTENT;
                                        }
                                        else
                                        {
                                            if(!(m_hDropTarget &&
m_rcDropTarget.PtInRect(pt)))
                                            {
                                                if(m_nDontDropCursor)
                                                    SetCursor
(AfxGetApp()->LoadCursor(m_nDontDropCursor));
                                                else
                                                    SetCursor
(AfxGetApp()->LoadStandardCursor(IDC_NO));
                                                m_iDropResult =
FHDR_DROPPED;
                                            }
                                            else
                                            {
                                                SetCursor(AfxGetApp()->
LoadStandardCursor(IDC_ARROW));
                                                m_iDropResult =
FHDR_ONTARGET;
                                            }
                                        }
                                    }
                                    if(iHotOrder == m_iHotOrder || iHotOrder
== m_iHotOrder+1)
                                        iHotOrder = -1;
                                    if(iHotOrder != m_iHotDivider)
                                        OnSetHotDivider(FALSE, iHotOrder);
                                }
                                return;
                            }
                            else if(GetStyle()&HDS_DRAGDROP)
                            {
                                INT iDragCX = GetSystemMetrics(SM_CXDRAG);
                                INT iDragCY = GetSystemMetrics(SM_CYDRAG);
                                CRect rectDrag(
                                    m_ptClickPoint.x-iDragCX,
m_ptClickPoint.y-iDragCY,
                                    m_ptClickPoint.x+iDragCX,
m_ptClickPoint.y+iDragCY
                                );
                                if(!rectDrag.PtInRect(point))
                                {
                                    NMHEADER nmhdr;
                                    nmhdr.hdr.hwndFrom = m_hWnd;
                                    nmhdr.hdr.idFrom = GetDlgCtrlID();
                                    nmhdr.hdr.code = HDN_BEGINDRAG;
                                    nmhdr.iItem = m_iHotIndex;
                                    nmhdr.iButton =   1;
                                    nmhdr.pitem = &m_hdiHotItem;

                                    BOOL bBeginDrag = TRUE;
```

```
                                    CWnd* pWnd = GetParent();
                                    if(pWnd != NULL)
                                          bBeginDrag = pWnd->SendMessage
(WM_NOTIFY, 0, (LPARAM)&nmhdr)==FALSE ? TRUE:FALSE;

                                    if(bBeginDrag)
                                    {
                                          ASSERT(m_pDragWnd == NULL);
                                          m_pDragWnd = new CFHDragWnd;
                                          if(m_pDragWnd)
                                          {
                                                CRect rectItem;
                                                VERIFY(GetItemRect
(m_iHotIndex, rectItem));
                                                ClientToScreen(&rectItem);

                                                m_pDragWnd->Create(rectItem,
this, m_iHotIndex, &m_hdiHotItem);
                                          }

                                          BOOL bVisible = FALSE;
                                          if(m_hDropTarget != NULL)
                                          {
                                                bVisible = ::GetWindowLong
(m_hDropTarget, GWL_STYLE)&WS_VISIBLE ? TRUE:FALSE;

                                                HWND hParent = ::GetParent
(m_hDropTarget);
                                                if(hParent)
                                                      bVisible =
::GetWindowLong(hParent, GWL_STYLE)&WS_VISIBLE ? TRUE:FALSE;
                                          }

                                          if(m_hDropTarget != NULL &&
bVisible)
                                                VERIFY(::GetWindowRect
(m_hDropTarget, m_rcDropTarget));
                                          else
                                                m_rcDropTarget.SetRectEmpty();
                                    }

                                    SetCapture();
                                    m_bDragging = TRUE;
                              }
                        }
                  }
            }
      }
```

```
void CFlatHeaderCtrl::OnCancelMode()
{
      CWnd::OnCancelMode();

      if(m_bDragging)
      {
            m_nClickFlags = 0;

            if(m_pDragWnd != NULL)
            {
                  m_pDragWnd->DestroyWindow();
                  m_pDragWnd = NULL;
```

```
            }

            if (GetCapture()->GetSafeHwnd() == GetSafeHwnd())
                  ReleaseCapture();

            m_bDragging = FALSE;
            OnSetHotDivider(FALSE, -1);

            Invalidate();
      }
}
```

```
////////////////////////////////////////////////////////////////////////
////
//    File:       CFlatHeaderCtrl.h
//    Version:    1.0.6
//
//    Author:           Maarten Hoeben
//    E-mail:           hoeben@nwn.com
//
//    Specification of the CFlatHeaderCtrl and associated classes.
//
//    This code may be used in compiled form in any way you desire. This
//    file may be redistributed unmodified by any means PROVIDING it is
//    not sold for profit without the authors written consent, and
//    providing that this notice and the authors name and all copyright
//    notices remains intact.
//
//    An email letting me know how you are using it would be nice as
well.
//
//    This file is provided "as is" with no expressed or implied
warranty.
//    The author accepts no liability for any damage/loss of business
that
//    this product may cause.
//
////////////////////////////////////////////////////////////////////////
////

#if !defined(AFX_FLATHEADERCTRL_H__2162BEB4_A882_11D2_B18A_B294B34D6940
__INCLUDED_)
#define AFX_FLATHEADERCTRL_H__2162BEB4_A882_11D2_B18A_B294B34D6940
__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FlatHeaderCtrl.h : header file
//

#include <afxtempl.h>
#include <tchar.h>

#define FLATHEADER_TEXT_MAX   80
#define FLATHEADER_TT_MAGIC   -13111971

////////////////////////////////////////////////////////////////////////
/////
// CFlatHeaderCtrl window

class CFlatHeaderCtrl;
class CFHDragWnd;
class CFHDropWnd;

// FlatHeader properties
#define FH_PROPERTY_SPACING                1
#define FH_PROPERTY_ARROW                  2
#define FH_PROPERTY_STATICBORDER    3
#define FH_PROPERTY_DONTDROPCURSOR  4
#define FH_PROPERTY_DROPTARGET             5
#define FH_PROPERTY_ENABLETOOLTIPS  6
```

```
// FlatHeader drop result
#define FHDR_DROPPED     -1
#define FHDR_ONHEADER    0
#define FHDR_ONTARGET    1
#define FHDR_PERSISTENT 2

// Extended header styles
#define HDF_EX_AUTOWIDTH       0x0001
#define HDF_EX_INCLUDESORT     0x0002
#define HDF_EX_FIXEDWIDTH      0x0004
#define HDF_EX_TOOLTIP         0x0008
#define HDF_EX_PERSISTENT      0x0010

typedef struct _HDITEMEX
{
      UINT  nStyle;
      INT          iMinWidth;
      INT          iMaxWidth;
      CString      strToolTip;

      _HDITEMEX() : nStyle(0), iMinWidth(0), iMaxWidth(-1) {};

} HDITEMEX, FAR* LPHDITEMEX;

class CFlatHeaderCtrl : public CHeaderCtrl
{
      friend class CFHDragWnd;

    DECLARE_DYNCREATE(CFlatHeaderCtrl)

// Construction
public:
      CFlatHeaderCtrl();

// Attributes
public:
      BOOL ModifyProperty(WPARAM wParam, LPARAM lParam);

      BOOL GetItemEx(INT iPos, HDITEMEX* phditemex) const;
      BOOL SetItemEx(INT iPos, HDITEMEX* phditemex);

      INT GetItemWidth(LPHDITEM lphdi, BOOL bIncludeSort = FALSE);

      void SetSortColumn(INT iPos, BOOL bSortAscending);
      INT GetSortColumn(BOOL* pbSortAscending = NULL);

      INT GetDropResult();

      INT GetHeight();
      void SetHeight(INT iHeight);

// Overrides
public:
      virtual ~CFlatHeaderCtrl();

      virtual void DrawItem(LPDRAWITEMSTRUCT);
      virtual void DrawItem(CDC* pDC, CRect rect, LPHDITEM lphdi, BOOL
bSort, BOOL bSortAscending);

      virtual int OnToolHitTest(CPoint point, TOOLINFO* pTI) const;
```

```
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFlatHeaderCtrl)
        //}}AFX_VIRTUAL

    // Implementation
    protected:
        BOOL m_bDoubleBuffer;
        INT m_iSpacing;
        SIZE m_sizeImage;
        SIZE m_sizeArrow;
        BOOL m_bStaticBorder;
        UINT m_nDontDropCursor;
        HWND m_hDropTarget;
        CRect m_rcDropTarget;
        INT m_iDropResult;

        INT m_iHotIndex;
        INT m_iHotOrder;
        BOOL m_bHotItemResizable;
        HDHITTESTINFO m_hdhtiHotItem;
        HDITEM m_hdiHotItem;
        HDITEMEX m_hdieHotItem;
        TCHAR m_szHotItemText[FLATHEADER_TEXT_MAX];

        BOOL m_bResizing;

        INT m_iHotDivider;
        COLORREF m_crHotDivider;
        CFHDropWnd* m_pDropWnd;

        BOOL m_bDragging;
        CFHDragWnd* m_pDragWnd;

        UINT m_nClickFlags;
        CPoint m_ptClickPoint;

        BOOL m_bSortAscending;
        INT m_iSortColumn;
        CArray<HDITEMEX, HDITEMEX> m_arrayHdrItemEx;

        COLORREF m_cr3DHighLight;
        COLORREF m_cr3DShadow;
        COLORREF m_cr3DFace;
        COLORREF m_crText;

        CFont m_font;

        INT m_iHeight;

        void DrawCtrl(CDC* pDC);
        INT DrawImage(CDC* pDC, CRect rect, LPHDITEM hdi, BOOL bRight);
        INT DrawBitmap(CDC* pDC, CRect rect, LPHDITEM hdi, CBitmap*
    pBitmap, BITMAP* pBitmapInfo, BOOL bRight);
        INT DrawText (CDC* pDC, CRect rect, LPHDITEM lphdi);
        INT DrawArrow(CDC* pDC, CRect rect, BOOL bSortAscending, BOOL
    bRight);

    // Generated message map functions
    protected:
        //{{AFX_MSG(CFlatHeaderCtrl)
        afx_msg LRESULT OnInsertItem(WPARAM wparam, LPARAM lparam);
```

```
        afx_msg LRESULT OnDeleteItem(WPARAM wparam, LPARAM lparam);
        afx_msg LRESULT OnSetImageList(WPARAM wparam, LPARAM lparam);
        afx_msg LRESULT OnSetHotDivider(WPARAM wparam, LPARAM lparam);
        afx_msg LRESULT OnLayout(WPARAM wparam, LPARAM lparam);
        afx_msg UINT OnNcHitTest(CPoint point);
        afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
        afx_msg void OnPaint();
        afx_msg void OnSysColorChange();
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnCancelMode();
        //}}AFX_MSG
      afx_msg LRESULT OnSetFont(WPARAM wParam, LPARAM lParam);
      afx_msg LRESULT OnGetFont(WPARAM wParam, LPARAM lParam);
        DECLARE_MESSAGE_MAP()

        BOOL OnToolTipNotify(UINT nId, NMHDR *pNMHDR, LRESULT *pResult);
};

/////////////////////////////////////////////////////////////////////////////
/////
// CFHDragWnd window

#define FHDRAGWND_CLASSNAME    _T("MFCFHDragWnd")

class CFHDragWnd : public CWnd
{
// Construction
public:
      CFHDragWnd();

// Attributes
public:

// Operations
public:

// Overrrides
protected:
    // Drawing
    virtual void OnDraw(CDC* pDC);

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CFHDragWnd)
      protected:
      virtual void PostNcDestroy();
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CFHDragWnd();
      virtual BOOL Create(CRect rect, CFlatHeaderCtrl* pFlatHeaderCtrl,
INT iItem, LPHDITEM lphdiItem);

protected:
      CFlatHeaderCtrl* m_pFlatHeaderCtrl;
      INT m_iItem;
      LPHDITEM m_lphdiItem;
```

```
        // Generated message map functions
protected:
        //{{AFX_MSG(CFHDragWnd)
        afx_msg void OnPaint();
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
////
// CFHDropWnd window

#define FHDROPWND_CLASSNAME    _T("MFCFHDropWnd")

class CFHDropWnd : public CWnd
{
// Construction
public:
        CFHDropWnd(COLORREF crColor);

// Attributes
public:

// Operations
public:
        void SetWindowPos(INT x, INT y);

// Overrrides
protected:
    // Drawing

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFHDropWnd)
        protected:
        virtual void PostNcDestroy();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CFHDropWnd();
        virtual BOOL Create(INT iHeight);

protected:
        CBrush m_brush;
        CRgn m_rgn;

        INT m_iHeight;

        // Generated message map functions
protected:
        //{{AFX_MSG(CFHDropWnd)
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
/////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_FLATHEADERCTRL_H__2162BEB4_A882_11D2
_B18A_B294B34D6940__INCLUDED_)
```

```
// FoldersView.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "FoldersView.h"
#include "NewFolderDialog.h"
#include <libxml/tree.h>
#include <vector>
#include "PlayerDoc.h"
#include "Windowsx.h" // for GET_X_LPARAM
#include "Utils.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CFoldersView

IMPLEMENT_DYNCREATE(CFoldersView, CTreeView)

CFoldersView::CFoldersView()
{
}

CFoldersView::~CFoldersView()
{
}


BEGIN_MESSAGE_MAP(CFoldersView, CTreeView)
      //{{AFX_MSG_MAP(CFoldersView)
      ON_COMMAND(ID_FILE_NEWFOLDER, OnFileNewfolder)
      ON_NOTIFY_REFLECT(TVN_SELCHANGED, OnSelchanged)
      ON_NOTIFY_REFLECT(NM_RCLICK, OnRclick)
      ON_COMMAND(ID_FOLDERS_ADDMEDIAITEMS, OnAddMediaToFolder)
      ON_COMMAND(ID_FOLDERS_RENAMEFOLDER, OnRenameFolderFromCM)
      ON_COMMAND(ID_FOLDERS_DELETEFOLDER, OnDeleteFolderFromCM)
      ON_NOTIFY_REFLECT(TVN_BEGINLABELEDIT, OnBeginlabeledit)
      ON_COMMAND(ID_FOLDERS_NEWFOLDER, OnNewFolderFromCM)
      ON_NOTIFY_REFLECT(TVN_ENDLABELEDIT, OnEndlabeledit)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CFoldersView drawing

void CFoldersView::OnDraw(CDC* pDC)
{
      CDocument* pDoc = GetDocument();
      // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
//////
// CFoldersView diagnostics
```

```
#ifdef _DEBUG
void CFoldersView::AssertValid() const
{
      CTreeView::AssertValid();
}

void CFoldersView::Dump(CDumpContext& dc) const
{
      CTreeView::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
/////
// CFoldersView message handlers

void CFoldersView::OnInitialUpdate()
{
      CTreeView::OnInitialUpdate();

      // TODO: Add your specialized code here and/or call the base class
      CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

      string cppstrLabel;

      xmlNodePtr node = pDoc->GetFolders();

      xmlNodePtr foldersnode = node;
      for(foldersnode;
            foldersnode != NULL;
            foldersnode = foldersnode->next) {
                  if(strcmp(reinterpret_cast<const char *>(node->name),
                        "Folders") == 0) {
                        AddFolderFromXML(foldersnode, NULL, true);
                  }
      }

      if(GetTreeCtrl().GetRootItem() != NULL)
            GetTreeCtrl().Expand(GetTreeCtrl().GetRootItem
(),TVE_EXPAND);

      SelectInbox();

      m_oleDropTarget.Register(this);
}

BOOL CFoldersView::PreCreateWindow(CREATESTRUCT& cs)
{
      // TODO: Add your specialized code here and/or call the base class
      cs.style |= TVS_HASLINES | TVS_LINESATROOT | TVS_HASBUTTONS |
            TVS_SHOWSELALWAYS | TVS_EDITLABELS;

      return CTreeView::PreCreateWindow(cs);
}

void CFoldersView::OnFileNewfolder()
{
      // TODO: Add your command handler code here
      CNewFolderDialog folderDlg;
```

```
	if (folderDlg.DoModal() != IDOK)
		return;

	CString fName;
	xmlNodePtr node;
	CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

	HTREEITEM hNewHTreeItem;

	fName = folderDlg.m_strNewFolderName;
	HTREEITEM hRoot = GetTreeCtrl().GetRootItem();

	node = pDoc->AddFolderToXML(fName.operator LPCTSTR());
	hNewHTreeItem = GetTreeCtrl().InsertItem((fName.operator LPCTSTR
()), hRoot);
	/*hNewTVItem.hItem = hNewHTReeItem;
	hNewTVItem.mask = TVIF_PARAM;
	hNewTVItem.lParam = (LPARAM)node;*/
	GetTreeCtrl().SetItemData(hNewHTreeItem, (DWORD)node);

}

HTREEITEM CFoldersView::AddFolderFromXML(xmlNodePtr node, HTREEITEM
hRoot, bool firstTime)
{
	// Pay attention.  This is a recursive function call.  It will add
all
	// the folders inside of "folder" too.
	// firstTime should be true if you call it from OnInitialUpdate
	// so it can set Inbox as the selected item.

	string strLabel;
	HTREEITEM tempTreeItem;
	xmlNodePtr folder = NULL;

	for(folder = node->children;
		folder != NULL;
		folder = folder->next) {
			if(strcmp(reinterpret_cast<const char *>
(folder->name),
				"Folder") == 0) {
				//if(DEBUG) std::cout << "DEBUG: Folder "
<< folder->name << endl;
				xmlChar *label;
				label = xmlGetProp(folder,
					reinterpret_cast<const unsigned char
*>
					("Label"));

				strLabel = reinterpret_cast<const char *>
(label);


				tempTreeItem = GetTreeCtrl().InsertItem
(strLabel.c_str(), hRoot);
				GetTreeCtrl().SetItemData(tempTreeItem,
(DWORD)folder);

				if((firstTime) && (strLabel == "Inbox")) {
					inboxTreeItem = tempTreeItem;
				}
```

```
                                    if(folder->children != NULL) {
                                          // If the folder contains folders,
do this again.
                                          AddFolderFromXML(folder,
tempTreeItem, firstTime);
                                    }

                              }
            }

            return tempTreeItem;
}

void CFoldersView::OnSelchanged(NMHDR* pNMHDR, LRESULT* pResult)
{
            NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
            // TODO: Add your control notification handler code here
            CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
            HTREEITEM selectedItem = GetTreeCtrl().GetSelectedItem();
            if(selectedItem == NULL) {
                  MessageBox("Huh?",
                        "Error",MB_OK);
            }

            xmlNodePtr folderNode;

            // Get the nodePtr attached to the folder to add to.
            folderNode = (xmlNodePtr) GetTreeCtrl().GetItemData(selectedItem);
            pDoc->SetCurSelectedFolder(folderNode);
            TRACE("Folder Selection Set.\n");
            *pResult = 0;
}

void CFoldersView::OnRclick(NMHDR* pNMHDR, LRESULT* pResult)
{
            CTreeCtrl & tc = GetTreeCtrl();

            /* Get the mouse cursor position */
            DWORD dwPos = GetMessagePos();

            /* Convert the co-ords into a CPoint structure */
            CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
            spt = pt;

            /* Convert to screen co-ords for hittesting */
            tc.ScreenToClient( &spt );
            //Set rightClickPoint for hittesting on the treeview.
            rightClickPoint = spt;

            // Add Popup menu
            CMenu menu;
            VERIFY(menu.LoadMenu(IDR_FOLDERS_MENU));
            CMenu* pPopup = menu.GetSubMenu(0);
            ASSERT(pPopup != NULL);

            pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, pt.x,
 pt.y, AfxGetMainWnd());

            *pResult = 0;
}
```

```
void CFoldersView::OnAddMediaToFolder() {
      CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

      UINT flags;
      HTREEITEM selectedItem = GetTreeCtrl().HitTest(rightClickPoint,
&flags);
      if(selectedItem == NULL) {
            MessageBox("You must have a folder selected before you can
add media.",
                  "Error",MB_OK);
            return;
      }

      // Get the nodePtr attached to the folder to add to.
      xmlNodePtr parentNode = (xmlNodePtr) GetTreeCtrl().GetItemData
(selectedItem);
      if(parentNode == NULL) {
            MessageBox("Error adding to folder.",
                  "Error",MB_OK);
            return;
      }

      char filename[102400];
      filename[0] = '\0';
      OPENFILENAME ofn;
      memset(&ofn, 0, sizeof(OPENFILENAME));
      ofn.lStructSize = sizeof(OPENFILENAME);
      ofn.hwndOwner = AfxGetMainWnd()->m_hWnd;
      ofn.Flags = OFN_ALLOWMULTISELECT | OFN_EXPLORER;
      ofn.lpstrFile = (LPSTR)filename;
      ofn.nMaxFile = 102400;
      if(GetOpenFileName(&ofn) == 0) {
            // User canceled or closed the dialog box or an error
occurred.
            //if(DEBUG) std::cout << "DEBUG: No File selected.\n";
            return;
      };

      int i = 0;
      vector<string> filenames;
      for(string file = &filename[i];
      file.compare("");
      file = &filename[i]) {
            i += file.length() + 1;
            //if(DEBUG) std::cout << "DEBUG: selected " << file << "\n";
            filenames.insert(filenames.end(), file);
      }
      if(filenames.size() == 1) {
            //if(DEBUG) std::cout << "DEBUG: one selected\n";
      }
      else {
            //if(DEBUG) std::cout << "DEBUG: multiple selected\n";
            for(unsigned int i=1; i<filenames.size(); i++) {
                  string full = filenames[0];
                  full.append("\\");
                  full.append(filenames[i]);
                  filenames[i] = full;
            }
      }
```

```
	typedef vector<string>::iterator VI;
	VI vi;
	if(filenames.size() == 1) {
		vi = filenames.begin();
	}
	else {
		vi = filenames.begin() + 1;
	}
	for(; vi != filenames.end(); vi++) {
		pDoc->AddMediaToXML(parentNode,
					*(vi),
					"Local",
					"-",
					*(vi),
					CUtils::get_currdatetime(),
					"-",
					"local",
					"none");
	}
	pDoc->UpdateAllViews(NULL);
}

void CFoldersView::SelectInbox()
{
	if(inboxTreeItem != NULL) { // will never happen. Just being safe.
		GetTreeCtrl().SelectItem(inboxTreeItem);
	}
}


void CFoldersView::OnDeleteFolderFromCM()
{
	CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

	UINT flags;
	HTREEITEM selectedItem = GetTreeCtrl().HitTest(rightClickPoint,
&flags);
	if(selectedItem == NULL) {
		MessageBox("You must click on a valid folder before you can
delete a folder.",
			"Error",MB_OK | MB_ICONEXCLAMATION);
		return;
	}

	xmlNodePtr curNode;

	// Get the nodePtr attached to the folder to add to.
	curNode = (xmlNodePtr) GetTreeCtrl().GetItemData(selectedItem);

	if((curNode == pDoc->GetInboxNodePtr()) || (curNode == pDoc->
GetTrashNodePtr())) {
		MessageBox("You can not delete the Inbox or the Trash.",
			"Error",MB_OK | MB_ICONEXCLAMATION);
		return;
	}

	if(MessageBox("Are you sure you want to permenantly delete this
folder and all items in it?  This will remove these files from your hard
drive.",
		"Confirm Folder Deletion",
		MB_YESNO | MB_ICONEXCLAMATION) == IDNO) return;
```

```
        SelectInbox();
        // Remove the folder the user right clicked on.
        GetTreeCtrl().DeleteItem(selectedItem);
        // Delete it from the xml structure.
        pDoc->DeleteFolder(curNode);

}

void CFoldersView::OnRenameFolderFromCM()
{
        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

        UINT flags;
        HTREEITEM selectedItem = GetTreeCtrl().HitTest(rightClickPoint,
&flags);
        if(selectedItem == NULL) {
                MessageBox("You must select a valid folder.",
                        "Error",MB_OK | MB_ICONEXCLAMATION);
                return;
        }

        GetTreeCtrl().EditLabel(selectedItem);
}

void CFoldersView::OnNewFolderFromCM()
{
        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

        UINT flags;
        HTREEITEM selectedItem = GetTreeCtrl().HitTest(rightClickPoint,
&flags);
        if(selectedItem == NULL) {
                MessageBox("You must click on a valid folder before you can
add new media.",
                        "Error",MB_OK | MB_ICONEXCLAMATION);
                return;
        }


        xmlNodePtr parentNode;

        // Get the nodePtr attached to the folder to add to.
        parentNode = (xmlNodePtr) GetTreeCtrl().GetItemData(selectedItem);

        if((parentNode == pDoc->GetInboxNodePtr()) || (parentNode ==
pDoc->GetTrashNodePtr())) {
                MessageBox("You can not add a sub folder to the Inbox or the
Trash.",
                        "Error",MB_OK | MB_ICONEXCLAMATION);
                return;
        }


        CNewFolderDialog folderDlg;

        if (folderDlg.DoModal() != IDOK)
                return;

        CString fName;
```

```
        xmlNodePtr node;

        HTREEITEM hNewHTreeItem;

        fName = folderDlg.m_strNewFolderName;

        node = pDoc->AddFolderToXML(fName.operator LPCTSTR(), parentNode);
        hNewHTreeItem = GetTreeCtrl().InsertItem((fName.operator LPCTSTR
()), selectedItem);
        GetTreeCtrl().SetItemData(hNewHTreeItem, (DWORD)node);

}

void CFoldersView::OnBeginlabeledit(NMHDR* pNMHDR, LRESULT* pResult)
{
        TV_DISPINFO* pTVDispInfo = (TV_DISPINFO*)pNMHDR;
        // TODO: Add your control notification handler code here
        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());


        if(pDoc->InInbox() || pDoc->InTrash()) {
                *pResult = 1;
        }
        else {
                *pResult = 0;
        }
}

void CFoldersView::OnEndlabeledit(NMHDR* pNMHDR, LRESULT* pResult)
{

        TV_DISPINFO* pTVDispInfo = (TV_DISPINFO*)pNMHDR;
        // TODO: Add your control notification handler code here

        if(pTVDispInfo->item.pszText == NULL) return;

        string newName;
        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());
        HTREEITEM curSelTreeItem = GetTreeCtrl().GetSelectedItem();
        GetTreeCtrl().SetItemText(curSelTreeItem,pTVDispInfo->
item.pszText);
        newName = pTVDispInfo->item.pszText;
        xmlNodePtr curNode = pDoc->GetCurSelectedFolder();

        xmlSetProp(curNode,
                reinterpret_cast<const unsigned char *>("Label"),
                reinterpret_cast<const unsigned char *>(newName.c_str()));
        *pResult = 0;
}

DROPEFFECT CFoldersView::OnDragEnter(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point)
{
    CTreeView::OnDragEnter (pDataObject, dwKeyState, point);

        return DROPEFFECT_NONE;
}

DROPEFFECT CFoldersView::OnDragOver(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point)
{
```

```
    CTreeView::OnDragOver (pDataObject, dwKeyState, point);

      // Verify a folder is available to drop into.
      /*-----------------------------------------------------
      -BEGIN- Find the folder to add to
      -----------------------------------------------------*/
      /* Get the mouse cursor position */
      DWORD dwPos = GetMessagePos();

      /* Convert the co-ords into a CPoint structure */
      CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
      spt = pt;

      /* Convert to screen co-ords for hittesting */
      GetTreeCtrl().ScreenToClient( &spt );

      UINT flags;
      HTREEITEM selectedItem = GetTreeCtrl().HitTest(spt, &flags);
      if(selectedItem == NULL) return DROPEFFECT_NONE;

      // Get the nodePtr attached to the folder to add to.
      xmlNodePtr parentNode =
            (xmlNodePtr) GetTreeCtrl().GetItemData(selectedItem);
      if(parentNode == NULL) return DROPEFFECT_NONE;
      /*-----------------------------------------------------
      -END- Find the folder to add to
      -----------------------------------------------------*/

      // Verify the data is in an acceptable format.
      if (pDataObject->IsDataAvailable(((CPlayerApp*) AfxGetApp ())->
GetClipboardFormat ())) {
            return (dwKeyState & MK_CONTROL) ?
            DROPEFFECT_COPY : DROPEFFECT_MOVE;
      }
    else if (pDataObject->IsDataAvailable(CF_HDROP))  {
            // Only Copy allowed on filenames
            return DROPEFFECT_COPY;
      }
      else
            return DROPEFFECT_NONE;
}

BOOL CFoldersView::OnDrop(COleDataObject* pDataObject, DROPEFFECT
dropEffect, CPoint point)
{
      CTreeView::OnDrop(pDataObject, dropEffect, point);

      /*-----------------------------------------------------
      -BEGIN- Find the folder to add to
      -----------------------------------------------------*/
      /* Get the mouse cursor position */
      DWORD dwPos = GetMessagePos();

      /* Convert the co-ords into a CPoint structure */
      CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
      spt = pt;

      /* Convert to screen co-ords for hittesting */
      GetTreeCtrl().ScreenToClient( &spt );

      UINT flags;
```

```
        HTREEITEM selectedItem = GetTreeCtrl().HitTest(spt, &flags);
        if(selectedItem == NULL) return false;

        // Get the nodePtr attached to the folder to add to.
        xmlNodePtr parentNode =
              (xmlNodePtr) GetTreeCtrl().GetItemData(selectedItem);
        if(parentNode == NULL) return false;
        /*-----------------------------------------------------
        -END- Find the folder to add to
        -----------------------------------------------------*/

        CPlayerDoc* pDoc = reinterpret_cast<CPlayerDoc *>(GetDocument());

        // See if an Episode is on the clipboard.
        UINT nFormat = ((CPlayerApp*) AfxGetApp ())->GetClipboardFormat
();
        HGLOBAL hData = pDataObject->GetGlobalData (nFormat);
        if (hData != NULL) {
              typedef struct { xmlNodePtr x; } S;
              S* s = (S*) ::GlobalLock (hData);
              xmlNodePtr p = s->x;
              pDoc->MoveNodeToFolderNode(p, parentNode);
              ::GlobalUnlock(hData);
              ::GlobalFree (hData);
              return TRUE; // Drop succeeded.
        }

        // See if bfilenames are on the clipboard.
      HDROP hDrop = (HDROP) pDataObject->GetGlobalData(CF_HDROP);
      if (hDrop != NULL) {
              // Find out how many file names the HDROP contains.
          int nCount = ::DragQueryFile (hDrop, (UINT) -1, NULL, 0);
          // Enumerate the file names.
          if (nCount) {
              TCHAR szFile[MAX_PATH];

                    for (int i=0; i<nCount; i++) {
                          ::DragQueryFile (hDrop, i, szFile,
                      sizeof (szFile) / sizeof (TCHAR));

                          pDoc->AddMediaToXML(parentNode,
                                                szFile,
                                                "Local",
                                                "-",
                                                szFile,

       CUtils::get_currdatetime(),
                                                "-",
                                                "local",
                                                "none");
                    }
                          pDoc->UpdateAllViews(NULL);
              }
              ::GlobalFree (hDrop);
              return TRUE; // Drop succeeded.
        }

      return FALSE;           // Drop failed.
  }
```

```
#if !defined(AFX_FOLDERSVIEW_H__D31A5359_A71E_4021_8712_
23131062A6FC__INCLUDED_)
#define AFX_FOLDERSVIEW_H__D31A5359_A71E_4021_8712_
23131062A6FC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#pragma warning(disable: 4786)
#endif // _MSC_VER > 1000

#include <libxml/tree.h>
#include <afxole.h>

/////////////////////////////////////////////////////////////////////////////
/////
// CFoldersView view

class CFoldersView : public CTreeView
{
protected:
      CFoldersView();           // protected constructor used by dynamic
creation
      DECLARE_DYNCREATE(CFoldersView)

// Attributes
public:

// Operations
public:
      void OnNewFolderFromCM();
      void SelectInbox();
      HTREEITEM AddFolderFromXML(xmlNodePtr node, HTREEITEM hRoot, bool
firstTime);

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CFoldersView)
      public:
      virtual void OnInitialUpdate();
      virtual DROPEFFECT OnDragEnter(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point);
      virtual DROPEFFECT OnDragOver(COleDataObject* pDataObject, DWORD
dwKeyState, CPoint point);
      virtual BOOL OnDrop(COleDataObject* pDataObject, DROPEFFECT
dropEffect, CPoint point);
      protected:
      virtual void OnDraw(CDC* pDC);      // overridden to draw this
view
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      //}}AFX_VIRTUAL

// Implementation
protected:
      virtual ~CFoldersView();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

      // Generated message map functions
protected:
```

```
        COleDropTarget m_oleDropTarget;

        HTREEITEM inboxTreeItem;
        CPoint rightClickPoint;

        //{{AFX_MSG(CFoldersView)
        afx_msg void OnFileNewfolder();
        afx_msg void OnSelchanged(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnRclick(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnAddMediaToFolder();
        afx_msg void OnRenameFolderFromCM();
        afx_msg void OnDeleteFolderFromCM();
        afx_msg void OnBeginlabeledit(NMHDR* pNMHDR, LRESULT* pResult);
        afx_msg void OnEndlabeledit(NMHDR* pNMHDR, LRESULT* pResult);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_FOLDERSVIEW_H__D31A5359_A71E_4021_8712_
23131062A6FC__INCLUDED_)
```

```
#if !defined(GLOBALS_H)
#define GLOBALS_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <string>
#include "stats.h"
#include <libxml/tree.h>
//#include "PlayerDoc.h"

#define DECLARE_USER_MESSAGE(name) \
      static const UINT name = ::RegisterWindowMessage(name##_MSG);

// Causes an UpdateAllViews() in the MainFrm
#define WM_USER_UPDATEALLVIEWS  (WM_APP + 1)
// Causes the MainFrm to update the progress bar
//#define WM_USER_SET_PROGRESS (WM_APP + 2)
#define WM_USER_SET_PROGRESS_MSG _T("WM_USER_SET_PROGRESS-9479553D-5D04-
4409-A938-0BC263AAD56C")
// Causes the Launch Media to be Enabled
#define WM_USER_WMDLG_CLOSE (WM_APP + 3)
// Causes media to be played.
#define WM_USER_PLAY (WM_APP + 4)
// Causes client to start downloaded content.  Used by the timer thread.
#define WM_USER_RECEIVE (WM_APP + 5)
// Tells the Client that the download thread is done.
#define WM_USER_DONE_DOWNLOAD (WM_APP + 6)
// Message from System Tray Icon
#define WM_USER_NOTIFYICON (WM_APP + 7)
// Load URL in Browser View
#define WM_USER_LOADURL (WM_APP + 8)
// Indicate a new Instance will be needed
#define WM_USER_DLDLG_CLOSE (WM_APP + 9)
// Systray switch icon
#define WM_USER_NOTIFYICON_ANIMATION_START (WM_APP + 10)
#define WM_USER_NOTIFYICON_ANIMATION_CYCLE (WM_APP + 11)
#define WM_USER_NOTIFYICON_ANIMATION_END (WM_APP + 12)
// Start syncronizer download.
#define WM_USER_STOP_DOWNLOAD (WM_APP + 14)
// Needed for sys tray icon animation.
#define NUM_ICONS_IN_DL_ANIMATION 6


static int iconResourceArray[NUM_ICONS_IN_DL_ANIMATION] =
{ IDI_ICON_DL_1, IDI_ICON_DL_2, IDI_ICON_DL_3, IDI_ICON_DL_4,
IDI_ICON_DL_5, IDI_ICON_DL_6 };


// IP Address and port of Server.
#define DEFAULT_IP _T("207.40.106.22")
#define DEFAULT_PORT _T("6666")

// Cookie path.
#define DEFAULT_COOKIE_PATH _T("/examples/servlet/")

#endif
```

```
// HardDriveFullDlg.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "HardDriveFullDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CHardDriveFullDlg dialog


CHardDriveFullDlg::CHardDriveFullDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CHardDriveFullDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CHardDriveFullDlg)
      m_icon = ::LoadIcon(NULL,IDI_HAND);
//    ::GetIcon(
      //}}AFX_DATA_INIT
}


void CHardDriveFullDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CHardDriveFullDlg)
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CHardDriveFullDlg, CDialog)
      //{{AFX_MSG_MAP(CHardDriveFullDlg)
      ON_BN_CLICKED(IDC_SUGGESTIONS, OnSuggestions)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CHardDriveFullDlg message handlers

void CHardDriveFullDlg::OnSuggestions()
{
      WinHelp(NULL);
      OnCancel();
}
```

```
#if !defined(AFX_HARDDRIVEFULLDLG_H__8C24087D_F672_4ECA_9BB5_
7B6387610962__INCLUDED_)
#define AFX_HARDDRIVEFULLDLG_H__8C24087D_F672_4ECA_9BB5_7B6387610962
__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// HardDriveFullDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
/////
// CHardDriveFullDlg dialog

class CHardDriveFullDlg : public CDialog
{
// Construction
public:
      CHardDriveFullDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CHardDriveFullDlg)
      enum { IDD = IDD_HARDDRIVEFULL };
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CHardDriveFullDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CHardDriveFullDlg)
      afx_msg void OnSuggestions();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
private:
      HICON m_icon;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_HARDDRIVEFULLDLG_H__8C24087D_F672_4ECA_9BB5_
7B6387610962__INCLUDED_)
```

```
#include "stdafx.h"
#include "ItemData.h"
#include <sac.h>

static const GUID guidCertInfoEx =
  { 0xc39bf696, 0xb776, 0x459c, { 0xa1, 0x3a, 0x4b, 0x71, 0x16, 0xab,
0x9f, 0x9 } };

static const BYTE bCertInfoEx_App[] =
  { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09 };

static const BYTE bCertInfoEx_SP[] =
  { 0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00,
    0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00 };

typedef struct {
  HRESULT hr;
  DWORD cbCert;
  BYTE pbCert[1];
} CERTINFOEX;

CItemData::CItemData() {
  m_fIsDevice = true;

  // Shared Device/Storage Members
  m_pStorageGlobals = NULL;
  m_pEnumStorage = NULL;
  m_szName[0] = 0;

  // Device-Only Memebers
  m_pDevice = NULL;
  m_pRootStorage = NULL;
  m_dwType = 0;
  FillMemory((void*)&m_SerialNumber, sizeof(m_SerialNumber), 0);
  m_szMfr[0] = 0;
  m_dwVersion = 0;
  m_dwPowerSource = 0;
  m_dwPercentRemaining = 0;
  m_hIcon = NULL;
  m_dwMemSizeKB = 0;
  m_dwMemBadKB = 0;
  m_dwMemFreeKB = 0;
  m_fExtraCertified = false;

  // Storage-Only Memebers
  m_pStorage = NULL;
  m_dwAttributes = 0;
  FillMemory((void*)&m_Format, sizeof(m_Format), 0);
  FillMemory((void*)&m_DateTime, sizeof(m_DateTime), 0);
  m_dwSizeLow = 0;
  m_dwSizeHigh = 0;
}

CItemData::~CItemData() {
  if(m_hIcon) {
    DestroyIcon(m_hIcon);
    m_hIcon = NULL;
  }

  if(m_pStorageGlobals) {
    m_pStorageGlobals->Release();
```

```
    m_pStorageGlobals = NULL;
  }
  if(m_pEnumStorage){
    m_pEnumStorage->Release();
    m_pEnumStorage = NULL;
  }
  if(m_pRootStorage) {
    m_pRootStorage->Release();
    m_pRootStorage = NULL;
  }
  if(m_pStorage) {
    m_pStorage->Release();
    m_pStorage = NULL;
  }
  if(m_pDevice) {
    m_pDevice->Release();
    m_pDevice = NULL;
  }
}

bool CItemData::Init(CWMDM* pWmdm) {
  m_pWmdm = pWmdm;

  return true;
}

bool CItemData::Init(IWMDMDevice* pDevice) {
  HRESULT hr;
  WCHAR wsz[MAX_PATH];
  ULONG ulFetched;

  // This is a device object
  m_fIsDevice = true;

  // Shared Device/Storage Memebers

  // Get the RootStorage, StorageGlobals and EnumStorage interfaces
  m_pRootStorage = NULL;
  m_pEnumStorage = NULL;
  m_pStorageGlobals = NULL;
  {
    IWMDMEnumStorage* pEnumRootStorage;

    hr = pDevice->EnumStorage(&pEnumRootStorage);
    if(hr != S_OK) return false;

    hr = pEnumRootStorage->Next(1, &m_pRootStorage, &ulFetched);
    if(hr != S_OK) return false;

    hr = m_pRootStorage->GetStorageGlobals(&m_pStorageGlobals);
    if(hr != S_OK) return false;

    hr = m_pRootStorage->EnumStorage(&m_pEnumStorage);
    if(hr != S_OK) return false;

    pEnumRootStorage->Release();
  }

  // Get Device Name
  hr = pDevice->GetName(wsz, sizeof(wsz)/sizeof(WCHAR) - 1);
  if(hr != S_OK) lstrcpy(m_szName, "");
```

```
  else WideCharToMultiByte(CP_ACP, 0L, wsz, -1, m_szName, sizeof
(m_szName),
                    NULL, NULL);

  // Device-Only Members

  // Set the Device Pointer and addref it
  m_pDevice = pDevice;
  m_pDevice->AddRef();

  // Get Device Type
  hr = pDevice->GetType(&m_dwType);
  if(hr != S_OK) m_dwType = 0L;

  // Get Device Serial Number
  BYTE abMAC[SAC_MAC_LEN];
  BYTE abMACVerify[SAC_MAC_LEN];
  HMAC hMAC;

  hr = pDevice->GetSerialNumber(&m_SerialNumber, (BYTE*)abMAC);
  if(hr == S_OK) {
    m_pWmdm->m_pSAC->MACInit(&hMAC);
    m_pWmdm->m_pSAC->MACUpdate(hMAC,
                        (BYTE*)(&m_SerialNumber),
                        sizeof(m_SerialNumber));
    m_pWmdm->m_pSAC->MACFinal(hMAC, (BYTE*)abMACVerify);
    if(memcmp(abMACVerify, abMAC, sizeof(abMAC)) != 0)
      FillMemory((void*)&m_SerialNumber, sizeof(m_SerialNumber), 0);
  }

  // Get Device Manufacturer
  hr = pDevice->GetManufacturer(wsz, sizeof(wsz)/sizeof(WCHAR) - 1);
  if(hr != S_OK) lstrcpy(m_szMfr,"");
  else WideCharToMultiByte(CP_ACP, 0L, wsz, -1, m_szMfr, sizeof
(m_szMfr),
                    NULL, NULL);

  // Get Device Version
  hr = pDevice->GetVersion(&m_dwVersion);
  if(hr != S_OK) m_dwVersion = 0xffff;

  GetPower();

  // Get Device Icon
  hr = pDevice->GetDeviceIcon((ULONG*)&m_hIcon);
  if(hr != S_OK) m_hIcon = NULL;

  GetSpace();

  // Call opaque command to exchange extended authentication info
  {
    HMAC hMAC;
    OPAQUECOMMAND Command;
    CERTINFOEX* pCertInfoEx;
    DWORD cbData_App = sizeof(bCertInfoEx_App)/sizeof(bCertInfoEx_App
[0]);
    DWORD cbData_SP = sizeof(bCertInfoEx_SP)/sizeof(bCertInfoEx_SP[0]);
    DWORD cbData_Send = sizeof(CERTINFOEX) + cbData_App;

    // Fill out opaque command structure
    memcpy(&(Command.guidCommand), &guidCertInfoEx, sizeof(GUID));
```

```
    Command.pData = (BYTE*)CoTaskMemAlloc(cbData_Send);
    if(!Command.pData) return false;

    Command.dwDataLen = cbData_Send;

    // Map the data in the opaque command to a CERTINFOEX structure, and
    // fill in the cert info to send
    pCertInfoEx = (CERTINFOEX*)Command.pData;
    pCertInfoEx->hr = S_OK;
    pCertInfoEx->cbCert = cbData_App;
    memcpy(pCertInfoEx->pbCert, bCertInfoEx_App, cbData_App);

    // Compute MAC
    m_pWmdm->m_pSAC->MACInit(&hMAC);
    m_pWmdm->m_pSAC->MACUpdate(hMAC, (BYTE*)(&(Command.guidCommand)),
                               sizeof(GUID));
    m_pWmdm->m_pSAC->MACUpdate(hMAC, (BYTE*)(&(Command.dwDataLen)),
                               sizeof(Command.dwDataLen));
    if(Command.pData)
      m_pWmdm->m_pSAC->MACUpdate(hMAC, Command.pData,
Command.dwDataLen);
    m_pWmdm->m_pSAC->MACFinal(hMAC, Command.abMAC);

    // Send the command
    hr = pDevice->SendOpaqueCommand(&Command);
    if(hr == S_OK) {
      BYTE abMACVerify[WMDM_MAC_LENGTH];

      // Compute MAC
      m_pWmdm->m_pSAC->MACInit(&hMAC);
      m_pWmdm->m_pSAC->MACUpdate(hMAC, (BYTE*)(&(Command.guidCommand)),
sizeof(GUID));
      m_pWmdm->m_pSAC->MACUpdate(hMAC, (BYTE*)(&(Command.dwDataLen)),
                               sizeof(Command.dwDataLen));
      if(Command.pData)
      m_pWmdm->m_pSAC->MACUpdate(hMAC, Command.pData,
Command.dwDataLen);
      m_pWmdm->m_pSAC->MACFinal(hMAC, abMACVerify);

      // Verify MAC matches
      if(memcmp(abMACVerify, Command.abMAC, WMDM_MAC_LENGTH) == 0) {
      // Map the data in the opaque command to a CERTINFOEX structure
      pCertInfoEx = (CERTINFOEX*)Command.pData;
      // In this simple extended authentication scheme, the callee must
      // provice the exact cert info
      if((pCertInfoEx->cbCert != cbData_SP) ||
         (memcmp(pCertInfoEx->pbCert, bCertInfoEx_SP, cbData_SP) == 0))
        m_fExtraCertified = true;
      }
    }
    if(Command.pData) CoTaskMemFree(Command.pData);
  }

  // Storage-Only Members (pointers/handles only)
  m_pStorage = NULL;

  return S_OK;
}

bool CItemData::Init(IWMDMStorage* pStorage) {
  HRESULT hr;
```

```
WCHAR wsz[MAX_PATH];

// This is a storage object
m_fIsDevice = false;

// Shared device/storage members

// Get a pointer to the StorageGlobals interface
hr = pStorage->GetStorageGlobals(&m_pStorageGlobals);
if(hr != S_OK) return false;

// Get the storage attributes
hr = pStorage->GetAttributes(&m_dwAttributes, &m_Format);
if(hr != S_OK) m_dwAttributes = 0;

// Get a pointer to the EnumStorage Interface
if(m_dwAttributes * WMDM_FILE_ATTR_FOLDER) {
  hr = pStorage->EnumStorage(&m_pEnumStorage);
  if(hr != S_OK) return false;
}
else m_pEnumStorage = NULL;

// Get the storage name
hr = pStorage->GetName(wsz, sizeof(wsz)/sizeof(WCHAR) - 1);
if(hr != S_OK) lstrcpy(m_szName, "");
else WideCharToMultiByte(CP_ACP, 0L, wsz, -1, m_szName, sizeof
(m_szName),
                         NULL, NULL);

// Device-Only Members (pointers/handles only)
m_pDevice = NULL;
m_pRootStorage = NULL;
m_hIcon = NULL;
m_fExtraCertified = false;

// Storage-Only Members

// Save the WMDM storage pointer
m_pStorage = pStorage;
m_pStorage->AddRef();

// Get the storage date
hr = pStorage->GetDate(&m_DateTime);
if(hr != S_OK) FillMemory((void*)&m_DateTime, sizeof(m_DateTime), 0);

// If the storage is a file, get its size
// If the storage is a folder, set the size to zero
m_dwSizeLow = 0;
m_dwSizeHigh = 0;
if(!(m_dwAttributes & WMDM_FILE_ATTR_FOLDER))
  hr = pStorage->GetSize(&m_dwSizeLow, &m_dwSizeHigh);

return true;
}

void CItemData::GetSpace() {
  // Get the total, free and bad space on the storage
  HRESULT hr;
  DWORD dwLow;
  DWORD dwHigh;
```

```
  m_dwMemSizeKB = 0;
  hr = m_pStorageGlobals->GetTotalSize(&dwLow, &dwHigh);
  if(hr == S_OK) {
    INT64 nSize = ((INT64)dwHigh << 32 | (INT64)dwLow) >> 10;
    m_dwMemSizeKB = (DWORD)nSize;
  }

  m_dwMemBadKB = 0;
  hr = m_pStorageGlobals->GetTotalBad(&dwLow, &dwHigh);
  if(hr == S_OK) {
    INT64 nSize = ((INT64)dwHigh << 32 | (INT64)dwLow) >> 10;
    m_dwMemBadKB = (DWORD)nSize;
  }

  m_dwMemFreeKB = 0;
  hr = m_pStorageGlobals->GetTotalFree(&dwLow, &dwHigh);
  if(hr == S_OK) {
    INT64 nSize = ((INT64)dwHigh << 32 | (INT64)dwLow) >> 10;
    m_dwMemFreeKB = (DWORD)nSize;
  }
}

void CItemData::GetPower() {
  // Get power source and power remaining.
  HRESULT hr;

  hr = m_pDevice->GetPowerSource(&m_dwPowerSource,
&m_dwPercentRemaining);
  if(hr != S_OK) {
    m_dwPowerSource = 0;
    m_dwPercentRemaining = 0;
  }
}

HRESULT CItemData::Refresh() {
  if(!m_fIsDevice) return E_UNEXPECTED; // Only valid for a device

  GetPower();
  GetSpace();

  return S_OK;
}
```

```
#if !defined(AFX_ITEMDATA_H__722DB86F_B723_4531_8C94_9B042E87F2D4
__INCLUDED_)
#define AFX_ITEMDATA_H__722DB86F_B723_4531_8C94_9B042E87F2D4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "wmdm.h"
#include <mswmdm.h>

class CItemData {
public:
  CItemData();
  virtual ~CItemData();
  bool Init(IWMDMDevice* pDevice);
  bool Init(IWMDMStorage* pStorage);
  bool Init(CWMDM* pWmdm);
  HRESULT Refresh();

  bool m_fIsDevice; // Flag indicating a device or storage item

  // Shared Device/Storage Members
  IWMDMStorageGlobals* m_pStorageGlobals;
  IWMDMEnumStorage* m_pEnumStorage;
  char m_szName[MAX_PATH];
  CWMDM* m_pWmdm;

  // Device-Only Members
  IWMDMDevice* m_pDevice;
  IWMDMStorage* m_pRootStorage;
  DWORD m_dwType;
  WMDMID m_SerialNumber;
  CHAR m_szMfr[MAX_PATH];
  DWORD m_dwVersion;
  DWORD m_dwPowerSource;
  DWORD m_dwPercentRemaining;
  HICON m_hIcon;
  DWORD m_dwMemSizeKB;
  DWORD m_dwMemBadKB;
  DWORD m_dwMemFreeKB;
  bool m_fExtraCertified;

  // Storage-Only Members
  IWMDMStorage* m_pStorage;
  DWORD m_dwAttributes;
  _WAVEFORMATEX m_Format;
  WMDMDATETIME m_DateTime;
  DWORD m_dwSizeLow;
  DWORD m_dwSizeHigh;

private:
  void GetSpace();
  void GetPower();
};

#endif // !defined(AFX_ITEMDATA_H__722DB86F_B723_4531_8C94_9B042E87F2D4
__INCLUDED_)
```

```
// Login.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "LoginDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CLoginDlg dialog


CLoginDlg::CLoginDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CLoginDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CLoginDlg)
      m_username = _T("");
      m_password = _T("");
      //}}AFX_DATA_INIT
}


void CLoginDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CLoginDlg)
      DDX_Text(pDX, IDC_EDIT1, m_username);
      DDX_Text(pDX, IDC_EDIT2, m_password);
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CLoginDlg, CDialog)
      //{{AFX_MSG_MAP(CLoginDlg)
            // NOTE: the ClassWizard will add message map macros here
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CLoginDlg message handlers
```

```
#if !defined(AFX_LOGIN_H__FC744D64_0506_4E29_BDA4_E7C8BFA3AE91
__INCLUDED_)
#define AFX_LOGIN_H__FC744D64_0506_4E29_BDA4_E7C8BFA3AE91__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Login.h : header file
//

/////////////////////////////////////////////////////////////////////////////
/////
// CLoginDlg dialog

class CLoginDlg : public CDialog
{
// Construction
public:
      CLoginDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CLoginDlg)
      enum { IDD = IDD_LOGIN_DLG };
      CString     m_username;
      CString     m_password;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CLoginDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CLoginDlg)
            // NOTE: the ClassWizard will add member functions here
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_LOGIN_H__FC744D64_0506_4E29_BDA4_E7C8BFA3AE91
__INCLUDED_)
```

```
#include "stdafx.h"
#include "player.h"

#include "MainFrm.h"
#include "DisplayView.h"
#include "FoldersView.h"
#include "PlayerDoc.h"
#include "HardDriveFullDlg.h"
#include "globals.h"
#include "Windowsx.h" // for GET_X_LPARAM

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
/////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

DECLARE_USER_MESSAGE(WM_USER_SET_PROGRESS)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
      //{{AFX_MSG_MAP(CMainFrame)
      ON_WM_CREATE()
      ON_MESSAGE(WM_USER_UPDATEALLVIEWS, OnMyUpdateMessage)
      ON_REGISTERED_MESSAGE(WM_USER_SET_PROGRESS, OnSetProgress)
      ON_MESSAGE(WM_USER_RECEIVE, OnReceiveMessage)
      ON_MESSAGE(WM_USER_STOP_DOWNLOAD, OnStopDownload)
      ON_MESSAGE(WM_USER_DONE_DOWNLOAD, OnDoneDownload)
      ON_MESSAGE(WM_USER_NOTIFYICON, OnNotifyIcon)
      ON_MESSAGE(WM_USER_NOTIFYICON_ANIMATION_START,
OnNotifyIconAnimationStart)
      ON_MESSAGE(WM_USER_NOTIFYICON_ANIMATION_CYCLE,
OnNotifyIconAnimationCycle)
      ON_MESSAGE(WM_USER_NOTIFYICON_ANIMATION_END,
OnNotifyIconAnimationEnd)
      ON_WM_CLOSE()
      ON_MESSAGE(WM_USER_PLAY, OnPlay)
      ON_MESSAGE(WM_USER_LOADURL, OnUserLoadURL)
      ON_MESSAGE(WM_USER_WMDLG_CLOSE, OnWMDlgClose)
      ON_MESSAGE(WM_USER_DLDLG_CLOSE, OnDlDlgClose)
      ON_WM_TIMER()
      ON_COMMAND(ID_TOOLS_RECEIVE, OnToolsReceive)
      ON_COMMAND(ID_SYNC_STATUS, OnSyncStatus)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
      ID_SEPARATOR,           // status line indicator
      ID_INDICATOR_CAPS,
      ID_INDICATOR_NUM,
      ID_INDICATOR_SCRL,
};


void CALLBACK EXPORT DLTimerProc(
```

```
    HWND hWnd,      // handle of CWnd that called SetTimer
    UINT nMsg,      // WM_TIMER
    UINT nIDEvent,   // timer identification
    DWORD dwTime    // system time
    ) {

        ::PostMessage(hWnd, WM_USER_RECEIVE, (LPARAM)0, (WPARAM)0);
        TRACE("DL Timer callback called.\n");

}
/////////////////////////////////////////////////////////////////////////////
/////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        m_displayView = NULL;
        m_foldersView = NULL;
        m_webView = NULL;
        m_mviewer = NULL;
        m_syncStatusDlg = NULL;
        m_iconAnimationCount = 0;
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

//      if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
WS_VISIBLE | CBRS_TOP
//              | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) ||
//              !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
//      {
//              TRACE0("Failed to create toolbar\n");
//              return -1;      // fail to create
//      }

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
//      m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
//      EnableDocking(CBRS_ALIGN_ANY);
//      DockControlBar(&m_wndToolBar);

        // Add Notify Icon.
        NOTIFYICONDATA nid;
        nid.cbSize = sizeof(NOTIFYICONDATA);
        m_notifyIconWnd = m_hWnd;
```

```
        nid.hWnd = m_notifyIconWnd;
        nid.uID = 0;
        nid.uFlags = NIF_ICON | NIF_TIP | NIF_MESSAGE;
        nid.hIcon = LoadIcon(AfxGetInstanceHandle(),
              MAKEINTRESOURCE(IDR_NOTIFYICON));
        nid.uCallbackMessage = WM_USER_NOTIFYICON;

        CString toolTip("Pushcast Player");

        LPCTSTR lpszToolTip = toolTip.GetBuffer(toolTip.GetLength());
        lstrcpyn(nid.szTip,lpszToolTip,strlen(lpszToolTip)+1);


        Shell_NotifyIcon(NIM_ADD,&nid);
        // Setup autoDL timer to check every minute.
        SetTimer( 1, 60000, DLTimerProc);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CFrameWnd::PreCreateWindow(cs) )
              return FALSE;
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        /*    Remove the Document Name from the Title Bar
              See "Programming Windows with MFC 2nd Ed." by Jeff Prosise
              p574-575 */
        cs.style &= ~FWS_ADDTOTITLE;

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
/////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
/////
// CMainFrame message handlers

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                                CCreateContext*
pContext)
{
```

```
      // create a splitter with 3 row, 1 columns
      if (!m_wndSplitter1.CreateStatic(this, 2, 1))
      {
            TRACE0("Failed to CreateStaticSplitter\n");
            return FALSE;
      }

      // add the first splitter pane - the default view in column 0
      // Add the Web View
      if (!m_wndSplitter1.CreateView(1, 0,
            RUNTIME_CLASS(CWebView), CSize(0,0), pContext))
      {
            TRACE0("Failed to create CWebView\n");
            return FALSE;
      }

      // add the second splitter pane - which is a nested splitter with
1 rows
      if (!m_wndSplitter2.CreateStatic(
            &m_wndSplitter1,     // our parent window is the first
splitter
            1, 2,                // the new splitter is 1 rows, 2 column
            WS_CHILD | WS_VISIBLE | WS_BORDER,  // style, WS_BORDER is
needed
            m_wndSplitter1.IdFromRowCol(0, 0)
                  // new splitter is in the first row, 2nd column of
first splitter
         ))
      {
            TRACE0("Failed to create nested splitter\n");
            return FALSE;
      }

      // now create the two views inside the nested splitter
      // Create the List View first because it is needed by the tree
view.
      if (!m_wndSplitter2.CreateView(0, 1,
            RUNTIME_CLASS(CDisplayView), CSize(0, 250), pContext))
      {
            TRACE0("Failed to create third pane\n");
            return FALSE;
      }
      if (!m_wndSplitter2.CreateView(0, 0,
            RUNTIME_CLASS(CFoldersView), CSize(150, 250), pContext))
      {
            TRACE0("Failed to create second pane\n");
            return FALSE;
      }

      m_wndSplitter1.SetRowInfo(0, 150,0);

      // it all worked, we now have two splitter windows which contain
      //  three different views

      // nove get references to them.
      //m_playerView = reinterpret_cast<CPlayerView *>
(m_wndSplitter1.GetPane(1, 0));
      m_displayView = reinterpret_cast<CDisplayView *>
(m_wndSplitter2.GetPane(0, 1));
      m_foldersView = reinterpret_cast<CFoldersView *>
(m_wndSplitter2.GetPane(0, 0));
```

```
	m_webView = reinterpret_cast<CWebView *>(m_wndSplitter1.GetPane
(1,0));

	return TRUE;
}

LRESULT CMainFrame::OnMyUpdateMessage(UINT wParam, LONG lParam)
{
	GetPlayerDoc()->UpdateAllViews(NULL);
	return 0; // I handled this message
}

LRESULT CMainFrame::OnSetProgress(WPARAM wParam, LPARAM) {
	if(m_syncStatusDlg != NULL) {
		m_syncStatusDlg->m_progress_overall.SetPos((int)wParam);
	}

	return 0;
}

LRESULT CMainFrame::OnNotifyIcon(WPARAM wParam, LPARAM lParam)
{
	CString toolTip("Pushcast Player");
	LPCTSTR lpszToolTip = toolTip.GetBuffer(toolTip.GetLength());

	UINT uID;
    UINT uMouseMsg;

    uID = (UINT) wParam;
    uMouseMsg = (UINT) lParam;

	  if (uMouseMsg == WM_LBUTTONDBLCLK) {
	    switch (uID) {
		case 0:
			ShowWindow(SW_RESTORE);

		break;

	    default:
			TRACE("Unhandled Task Tray Left Click.\n");
		break;
	    }
	  }
	  if (uMouseMsg == WM_RBUTTONUP) {
		TRACE("WM_RBUTTONUP from systray icon.\n");
		switch(0) {
		case 0:
			TRACE("Right Click On Task Tray.\n");
			DoRightClickOnTrayIcon();
			break;
		default:
			TRACE("Unhandled WM_NOTIFY message.\n");
			break;
		}
	  }
	  if (uMouseMsg == WM_MOUSEMOVE) {
		switch (uID) {
		case IDI_NOTIFYICON :
//			char tip[64];
//			char val[32];
```

```
                    // Modify tooltip to contain current status.
                    NOTIFYICONDATA nid;
                    nid.cbSize = sizeof(NOTIFYICONDATA);
                    nid.hWnd = m_notifyIconWnd;
                    nid.uID = IDI_NOTIFYICON;
                    nid.uFlags = NIF_TIP;
/*                  if(m_playerView->m_progressCtrl.GetPos() == 0)
                          strcpy(tip, "Pushcast Player\nStatus: Not
Connected");
                    else
                    {
                          strcpy(tip, "Pushcast Player\nStatus:
Downloading\n(");
                          itoa(m_playerView->m_progressCtrl.GetPos
(),val,10);
                          strcat(tip, val);
                          strcat(tip, "% complete)");
                    }
*/                  //strcpy(nid.szTip, tip);

                    lstrcpyn(nid.szTip,lpszToolTip,strlen(lpszToolTip)+1);
                    Shell_NotifyIcon(NIM_MODIFY,&nid);
                    break;

              default:
                    break;
              }
        }

        return TRUE;
}

void CMainFrame::OnClose()
{
        // TODO: Add your message handler code here and/or call default
        // Empty trash.
        GetPlayerDoc()->EmptyTrash();

        // Stop the Download.
        GetPlayerDoc()->StopDownload();

        // Delete the Status Window if there is one.
        DeleteStatusDlg();

        // Delete the Media Window if one exists.
        DeleteMediaWindow();

        // Delete Notify Icon
        NOTIFYICONDATA nid;
        nid.hWnd = m_notifyIconWnd;
        nid.uID = IDI_NOTIFYICON;
        Shell_NotifyIcon(NIM_DELETE,&nid);

        CFrameWnd::OnClose();
}

void CMainFrame::DoRightClickOnTrayIcon()
{
        DWORD dwPos = GetMessagePos();

        /* Convert the co-ords into a CPoint structure */
```

```
	CPoint pt( GET_X_LPARAM( dwPos ), GET_Y_LPARAM( dwPos ) ), spt;
	spt = pt;

	/* Convert to screen co-ords for hittesting */
	//lc.ScreenToClient( &spt );
	// Set rightClickPoint incase someone needs it in the future
	// for hittesting.

	//rightClickPoint = spt;

	// Add Popup menu


	CMenu menu;
	VERIFY(menu.LoadMenu(IDR_TASKTRAY_MENU));
	CMenu* pPopup = menu.GetSubMenu(0);
	ASSERT(pPopup != NULL);
	::SetForegroundWindow(m_notifyIconWnd);

	pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, pt.x,
pt.y, AfxGetMainWnd());

	::PostMessage(m_notifyIconWnd,WM_NULL, 0, 0);
}

//void CMainFrame::OnTaskTrayAbout()
//{
//    TRACE("About From Task Tray Message.\n");
//}

void CMainFrame::OnUserLoadURL()
{
	CWebView *webView;
	webView = reinterpret_cast<CWebView *>(m_wndSplitter1.GetPane(1,
0));
	webView->go();
}

// Purpose: Show the Download Status Dialog.
void CMainFrame::showStatus()
{
	if(m_syncStatusDlg == NULL) {
		m_syncStatusDlg = new CSyncStatusDlg(NULL);
		m_syncStatusDlg->Create(IDD_SYNCSTATUS, this);
		if(GetPlayerDoc()->IsDownloading()) {
			m_syncStatusDlg->EnableCancelButton();
		}
	}
}

void CMainFrame::SetupIconAnimation(DWORD dwMsgType,UINT nIndexOfIcon,
CString strToolTip)
{
	//Load the specified icon from the array
	HICON hIconAtIndex = AfxGetApp()->LoadIcon(iconResourceArray
[nIndexOfIcon]);

	///Fill up the NOTIFYICONDATA Structure
	NOTIFYICONDATA iconData;
	iconData.cbSize          = sizeof(NOTIFYICONDATA);
	iconData.hWnd            = m_notifyIconWnd;  //window's handle
```

```
      iconData.uID                    = IDI_NOTIFYICON ;  //identifier
      iconData.uFlags          = NIF_MESSAGE|NIF_ICON|NIF_TIP; //flags
      iconData.uCallbackMessage       = WM_USER_NOTIFYICON; //notification
handler
      iconData.hIcon           = hIconAtIndex;     //icon handle

      //Fill up tool tip
      //LPCTSTR lpszToolTip = strToolTip.GetBuffer(strToolTip.GetLength
());
      //lstrcpyn(iconData.szTip,lpszToolTip,strlen(lpszToolTip)+1);

      //Tell the shell what we intend doing
      //Add,Delete,Modify ---> NIM_ADD,NIM_DELETE, NIM_MODIFY in
dwMsgType
      Shell_NotifyIcon(dwMsgType, &iconData);


      //Resources are precious !
      if (hIconAtIndex)
            DestroyIcon(hIconAtIndex);
}

void CMainFrame::CleanUpIconAnimation()
{

}

LRESULT CMainFrame::OnNotifyIconAnimationStart(WPARAM wParam, LPARAM
lParam) {
      int nIndexFirstIcon = 0;
      m_iconAnimationCount = 0;
    SetupIconAnimation(NIM_MODIFY,nIndexFirstIcon,"");
      m_iconAnimationCount++;

      SetTimer(2,1000,NULL);
      return 0;
}

LRESULT CMainFrame::OnNotifyIconAnimationCycle(WPARAM wParam, LPARAM
lParam) {

      SetupIconAnimation(NIM_MODIFY,m_iconAnimationCount,"");
      m_iconAnimationCount++;
      m_iconAnimationCount = m_iconAnimationCount%
(NUM_ICONS_IN_DL_ANIMATION);

      return 0;
}

LRESULT CMainFrame::OnNotifyIconAnimationEnd(WPARAM wParam, LPARAM
lParam) {
      m_iconAnimationCount = 0;
      NOTIFYICONDATA nid;
      nid.cbSize = sizeof(NOTIFYICONDATA);
      m_notifyIconWnd = m_hWnd;
      nid.hWnd = m_notifyIconWnd;
      nid.uID = IDI_NOTIFYICON;
      nid.uFlags = NIF_ICON | NIF_TIP | NIF_MESSAGE;
      nid.hIcon = LoadIcon(AfxGetInstanceHandle(),
            MAKEINTRESOURCE(IDR_NOTIFYICON));
```

```
	nid.uCallbackMessage = WM_USER_NOTIFYICON;
	strcpy(nid.szTip, _T("Pushcast Player"));

	KillTimer(2);

	Shell_NotifyIcon(NIM_MODIFY,&nid);

	return 0;
}

void CMainFrame::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	SetupIconAnimation(NIM_MODIFY,m_iconAnimationCount,"");
	m_iconAnimationCount++;
	m_iconAnimationCount = m_iconAnimationCount%
(NUM_ICONS_IN_DL_ANIMATION);

	CFrameWnd::OnTimer(nIDEvent);
}

//////////////////////////////////////////
//----------Functions-------------------//

// Purpose: Get the PlayerDocument
//                (eliminates the need to cast GetActiveDocument()).
CPlayerDoc* CMainFrame::GetPlayerDoc()
{
	return reinterpret_cast<CPlayerDoc*>(GetActiveDocument());
}

// Purpose:  Delete the Status Dilog if one exists.
void CMainFrame::DeleteStatusDlg()
{
	if(m_syncStatusDlg != NULL) {
		m_syncStatusDlg->DestroyWindow();
		delete m_syncStatusDlg;
		m_syncStatusDlg = NULL;
	}
}

// Purpose: Delete the Media Window if one exists.
void CMainFrame::DeleteMediaWindow()
{
	// Delete the media dialog if there is one.
	if(m_mviewer != NULL) {
//		m_mviewer->DestroyWindow();
		m_mviewer = NULL;
	}
}

//////////////////////////////////////////
//----------Menu Handlers---------------//
// Purpose: Send a message to syncronizer to download new media
//                when the ManulaReceive menu item is selected.
void CMainFrame::OnToolsReceive() {
	showStatus();
	StartDownload();
}

//////////////////////////////////////////
```

```
//-----------Message Handlers-------------//

// Purpose: When the WM_USER_PLAY message is received, either create a
//                new media dialog and play the media or play the media
//                in the existing dialog.
void CMainFrame::OnPlay()
{
      // Create a new Media Dialog if one is not created.
      if(m_mviewer == NULL) {
            CRuntimeClass* pRuntimeClass = RUNTIME_CLASS(CMediaFrame);
            m_mviewer = reinterpret_cast<CMediaFrame *>(pRuntimeClass->
CreateObject());
            // Set the Stats object in the Media Window object.
            m_mviewer->SetStats(GetPlayerDoc()->GetStats());
      }
      // else play the selected file.
      else {
            // Stop the currently playing file.
            m_mviewer->Stop();
      }

      // Play the currently selected file.
      m_mviewer->SetActiveWindow();
      m_mviewer->Play(GetPlayerDoc()->GetCurSelectedItem());
}

// Purpose: Message handler to delete the status object
//                when it is closed.
LRESULT CMainFrame::OnDlDlgClose(WPARAM wParam, LPARAM) {
      DeleteStatusDlg();

      return 0;
}

// Purpose:  Destroy the Media window and free the memory when it is
closed.
LRESULT CMainFrame::OnWMDlgClose(WPARAM wParam, LPARAM){
      DeleteMediaWindow();

      return 0;
}

void CMainFrame::OnReceiveMessage()
{
      m_elapsedMinutes++;
      if(m_elapsedMinutes > GetPlayerDoc()->GetDLInterval()) {
            m_elapsedMinutes = 0;
            StartDownload();
      }
      TRACE("Uped elapsed time by 1 minutes.\n");
}

// Purpose: Tell Document to stop downloading.
void CMainFrame::OnStopDownload()
{
      GetPlayerDoc()->StopDownload();
}

// Purpose: Cleans up the thread when it is done downloading.
void CMainFrame::OnDoneDownload(WPARAM wParam, LPARAM lParam) {
      TRACE("DownloadThread exit code: %d\n", lParam);
```

```
        // Check the return code.
        CHardDriveFullDlg* hdFullDlg = NULL;
        switch(lParam) {
        case DT_DISKSPACEERROR:
              // Show Error Dialog.
              hdFullDlg = new CHardDriveFullDlg();
              hdFullDlg->DoModal();
              delete hdFullDlg;
              break;
        default:
              break;
        }

        // Gray out Cancel on Sync status and the un gray the menu item
        if(m_syncStatusDlg) {
              m_syncStatusDlg->DisableCancelButton();
              DeleteStatusDlg();
        }
        GetPlayerDoc()->DownloadThreadCleanup();
}


void CMainFrame::StartDownload()
{
        GetPlayerDoc()->PurgeAndDownloadSubscriptions();
        if(m_syncStatusDlg) {
              m_syncStatusDlg->EnableCancelButton();
        }
}


// Purpose: Show the Sync Status Dialog.
void CMainFrame::OnSyncStatus()
{
        showStatus();
}
```

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////
/////

#if !defined(AFX_MAINFRM_H__51A27727_F808_49B2_9438_25C12ECC3778
__INCLUDED_)
#define AFX_MAINFRM_H__51A27727_F808_49B2_9438_25C12ECC3778__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "DisplayView.h"
#include "FoldersView.h"
#include "WebView.h"
#include "MediaFrame.h"

#define IDI_NOTIFYICON 0

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
      CMainFrame();
      DECLARE_DYNCREATE(CMainFrame)

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMainFrame)
      public:
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      //}}AFX_VIRTUAL

// Implementation
public:
      void StartDownload();
      int m_iconAnimationCount;
      CPlayerDoc* GetPlayerDoc();
      void CleanUpIconAnimation();
      void SetupIconAnimation(DWORD dwMsgType,UINT nIndexOfIcon, CString
strToolTip);
      void showStatus();
      virtual ~CMainFrame();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
      CStatusBar        m_wndStatusBar;
      CToolBar          m_wndToolBar;
      CSplitterWnd      m_wndSplitter1;
      CSplitterWnd      m_wndSplitter2;
      CFoldersView      *m_foldersView;
      CDisplayView      *m_displayView;
      CWebView          *m_webView;
```

```
// Generated message map functions
protected:
      void DoRightClickOnTrayIcon();

      virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs,
                  CCreateContext* pContext);
      //{{AFX_MSG(CMainFrame)
      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
      afx_msg LPARAM OnMyUpdateMessage(UINT wParam, LONG lParam);
      afx_msg LRESULT OnSetProgress(WPARAM wParam, LPARAM);
      afx_msg void OnReceiveMessage();
      afx_msg void OnSyncStatus();
      afx_msg LRESULT OnNotifyIcon(WPARAM wParam, LPARAM lParam);
      afx_msg LRESULT OnNotifyIconAnimationStart(WPARAM wParam, LPARAM
lParam);
      afx_msg LRESULT OnNotifyIconAnimationCycle(WPARAM wParam, LPARAM
lParam);
      afx_msg LRESULT OnNotifyIconAnimationEnd(WPARAM wParam, LPARAM
lParam);
      afx_msg void OnClose();
      afx_msg void OnPlay();
      afx_msg void OnUserLoadURL();
      afx_msg LRESULT OnWMDlgClose(WPARAM wParam, LPARAM);
      afx_msg LRESULT OnDlDlgClose(WPARAM wParam, LPARAM);
      afx_msg void OnTimer(UINT nIDEvent);
      afx_msg void OnToolsReceive();
      afx_msg void OnStopDownload();
      afx_msg void OnDoneDownload(WPARAM wParam, LPARAM lParam);
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
private:
      void DeleteMediaWindow();
      void DeleteStatusDlg();
//    CSyncThread *m_syncThread;
      CSyncStatusDlg* m_syncStatusDlg;
      CMediaFrame* m_mviewer;
      HWND m_notifyIconWnd;
      int m_elapsedMinutes;
};

/////////////////////////////////////////////////////////////////////////////
/////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_MAINFRM_H__51A27727_F808_49B2_9438_25C12ECC3778
__INCLUDED_)
```

```
// MediaFrame.cpp : implementation file
//

#include "stdafx.h"
#include "player.h"
#include "MediaFrame.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
/////
// CMediaFrame

IMPLEMENT_DYNCREATE(CMediaFrame, CFrameWnd)

CMediaFrame::CMediaFrame()
{
      RECT r;
    r.left = 100;
    r.top = 100;
    r.right = 200;
    r.bottom = 200;

      // Create the Media Window.
    Create(NULL, "Media Window",
           WS_SYSMENU|WS_THICKFRAME|WS_MINIMIZEBOX|WS_MAXIMIZEBOX,
        r, NULL, 0);

      // Create the MediaControl.
      m_mp.Create(NULL, NULL, WS_DLGFRAME, r, this, IDC_MEDIAPLAYER1,
NULL);
      m_mp.SetSendPlayStateChangeEvents(TRUE);
      m_mp.SetAutoSize(TRUE);
      m_mp.SetDisplaySize(0);
      m_mp.ShowWindow(SW_SHOW);
}

CMediaFrame::~CMediaFrame()
{
}

BEGIN_EVENTSINK_MAP(CMediaFrame, CFrameWnd)
    //{{AFX_EVENTSINK_MAP(CMediaFrame)
      ON_EVENT(CMediaFrame, IDC_MEDIAPLAYER1, 3012 /* PlayStateChange
*/, OnPlayStateChangeMediaplayer1, VTS_I4 VTS_I4)
      ON_EVENT(CMediaFrame, IDC_MEDIAPLAYER1, 2 /* PositionChange */,
OnPositionChangeMediaplayer1, VTS_R8 VTS_R8)
      ON_EVENT(CMediaFrame, IDC_MEDIAPLAYER1, 3002 /* EndOfStream */,
OnEndOfStreamMediaplayer1, VTS_I4)
      //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

BEGIN_MESSAGE_MAP(CMediaFrame, CFrameWnd)
      //{{AFX_MSG_MAP(CMediaFrame)
            // NOTE - the ClassWizard will add and remove mapping macros
here.
            ON_WM_CLOSE()
```

```
            ON_WM_SIZE()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
/////
// CMediaFrame message handlers

BOOL CMediaFrame::PreCreateWindow(CREATESTRUCT& cs)
{
      // TODO: Add your specialized code here and/or call the base class

      return CFrameWnd::PreCreateWindow(cs);
}

// Purpose:  Handles the stats logging when control buttons are pressed.
void CMediaFrame::OnPlayStateChangeMediaplayer1(long OldState, long
NewState)
{
      // Play was pushed.
      if(NewState == 2) {
            // Player was paused.
            if(OldState == 1) {
//                m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
//                                          0, 0, m_mp.GetCurrentPosition
(), "unpause");
            }
            // Player was stopped.
            else {
                m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
                                          0, 0, m_mp.GetCurrentPosition
(), "start");
            }
      }
      // Pause was pushed.
      else if(NewState == 1) {
//                m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
//                                          0, 0, m_mp.GetCurrentPosition
(), "pause");
      }
      // Stop was pushed.
      else if(NewState == 0) {
//                m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
//                                          0, 0, m_mp.GetCurrentPosition
(), "stop");
      }
}

// Purpose:  This is called before the window is closed.
//           Enables the Launch Media Button in the Player View.
void CMediaFrame::OnClose()
{
      // Media was playing when the window was closed so
      // record a stop stat.
      long playState = m_mp.GetPlayState();
      if(playState == 2) {
          m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
                                    0, 0, m_mp.GetCurrentPosition(),
"stop");
      }
```

```
        // Enable the Launch Media Button in the Player View.
        // By sending the WM_USER_WMDLG_CLOSE message to the playerview.
        ::PostMessage(AfxGetMainWnd()->m_hWnd,WM_USER_WMDLG_CLOSE,(WPARAM)
0,(LPARAM)0);

        CFrameWnd::OnClose();
}

// Purpose:  Called whenever the position of the media is changed.
void CMediaFrame::OnPositionChangeMediaplayer1(double oldPosition,
double newPosition)
{
        // When the new position is 0,
        if(newPosition == 0) {
                // stop has been pressed so log a stop stat.
                m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
                                        0, 0, oldPosition, "stop");
        }
}

// Purpose:  Resize the media dialog when the user resizes the dialog.
void CMediaFrame::OnSize(UINT nType, int cx, int cy)
{
        CFrameWnd::OnSize(nType, cx, cy);

        // Check for a valid media player object.
        if(m_mp) {
                // Resize the object to fit the window.
                m_mp.MoveWindow(0,0,cx,cy,TRUE);
        }
}

void CMediaFrame::OnEndOfStreamMediaplayer1(long Result)
{
        // The end has been reached so record a stop stat.
        m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
                                0, 0, m_mp.GetDuration(), "stop");

        xmlNodePtr toSetToCurrent = NULL;
        if(curPlayingNode == NULL) {return;}
        if(curPlayingNode->next == NULL) { return; }

        bool playNext = false;
        if(curPlayingNode->next != NULL) {
                if(curPlayingNode->next->type == XML_ELEMENT_NODE) {
                        toSetToCurrent = curPlayingNode->next;
                        playNext = true;

                }

        }
        else {
                return; // there isn't even a next node
        }
        if(curPlayingNode->next->next != NULL && playNext != true) {
                if(curPlayingNode->next->next->type == XML_ELEMENT_NODE) {
                        toSetToCurrent = curPlayingNode->next->next;
                        playNext = true;
                }
        }
        if(playNext) {
```

```
            curPlayingNode = toSetToCurrent;
            Play(curPlayingNode);
      }
}

// Purpose: Initialize the Player with the selected media
//                and start the media playing.
void CMediaFrame::Play(xmlNodePtr curItemNode) {
      // Get the selected Document.
      curPlayingNode = curItemNode;

      // If a Document is selected,
      if(curPlayingNode != NULL) {
            // Get the filename.
            xmlChar *url = xmlGetProp(curPlayingNode,
                  reinterpret_cast<const unsigned char *>
                  ("URL"));
            xmlChar *show = xmlGetProp(curPlayingNode,
                  reinterpret_cast<const unsigned char *>
                  ("SHOW"));

            xmlChar *desc = xmlGetProp(curPlayingNode,
                  reinterpret_cast<const unsigned char *>
                  ("DESC"));

            xmlSetProp(curPlayingNode,
                  reinterpret_cast<const unsigned char *>("LISTENED"),
                  reinterpret_cast<const unsigned char *>("1"));

            string strURL = reinterpret_cast<const char *>(url);
            string strDESC = reinterpret_cast<const char *>(desc);
            string strSHOW = reinterpret_cast<const char *>(show);
            string strTITLE = strSHOW + " - " + strDESC;
            // Open the media player with the file.
            m_mp.SetFileName(strURL.c_str());
            // Set the window title to the description.
            SetWindowText(strTITLE.c_str());
            // Set the window Icon to the parent icon.  This is a
temporary fix.
            SetIcon(LoadIcon(AfxGetInstanceHandle(), MAKEINTRESOURCE
(IDR_MAINFRAME)) ,false);

            // Resize dialog to size of media.
            CRect mediaRect;
            m_mp.GetWindowRect(mediaRect);

            CRect windowRect;
            GetWindowRect(windowRect);

            CRect clientRect;
            GetClientRect(clientRect);

            int deltaWidth = windowRect.Height() - clientRect.Height();
            int deltaHeight = windowRect.Height() - clientRect.Height();

            MoveWindow(windowRect.left,windowRect.top,deltaWidth +
mediaRect.Width(), deltaHeight + mediaRect.Height());
            ShowWindow(SW_SHOW);
            UpdateWindow();

            // Play the open the currently loaded media.
```

```
            m_mp.Play();
        }
}

// Purpose: Stop the currently playing media and sets the stat.
void CMediaFrame::Stop() {
        // Stop media.
        m_mp.Stop();
        // Add stop stat.
        m_stats->addStat(0, (LPCTSTR)m_mp.GetFileName(),
                                0, 0, m_mp.GetCurrentPosition(), "stop");
}

// Purpose: Set the local Stats object to the passed in Stats object.
void CMediaFrame::SetStats(CStats* s) {
        m_stats = s;
}
```